MSX2-BASIC

Albert Sickler

MSX2 BASIC

MET MONITORGEGEVENS

Albert Sickler

Kluwer Technische Boeken

MSX2-BASIC

met
monitorgegevens

Albert Sickler

MSX2-BASIC

met monitorgegevens

Kluwer Technische Boeken B.V.
Deventer – Antwerpen

ISBN 90 201 1961 3
D/1986/0108/220

1e druk 1986

WOORD VOORAF

Achter de drie letters MSX gaat een hele wereld schuil. Een wereld die wordt bepaald door een grote hoeveelheid verschillenden computermerken, die merkwaardigerwijze eerder opvallen door hun overeenkomst dan door hun verschil. Voor de eerste keer is in de wereld van de computers een *standaard* geïntroduceerd en de gevolgen zullen zonder twijfel enorm zijn.

Met de introductie van MSX-computers voor het grote publiek is een grote interesse ontstaan rond wat nu wel precies MSX-computers zijn en wat men met MSX-BASIC kan doen.

MSX2-BASIC is een nieuwe ontwikkeling in dit gebeuren. Terwijl de compatibiliteit met het bestaande MSX-BASIC gewaarborgd blijft, kent MSX2-BASIC vele nieuwe mogelijkheden, met name op het gebied van grafische en geluidstoepassingen. De MSX2-standaard gaat bovendien uit van een configuratie met ingebouwde diskdrive en een 80-koloms beeldscherm, zodat duidelijk sprake is van professionelere mogelijkheden.

Voor diegenen, die al kennis hebben van Z80-assembleertaal, is er bovendien een appendix opgenomen met gegevens over het gebruik van de ingebouwde BIOS-routines.

INHOUD

INLEIDING IN BASIC

1 DE EERSTE STAPPEN: OVER PRINT EN REKENKUNDIGE BEWERKINGEN

PRINT voor het uitvoeren van directe berekeningen

Als we de computer aanzetten, wordt deze onmiddellijk in de toestand gebracht waarin we met BASIC kunnen werken.

Om dit te tonen, typen we in:

```
PRINT 2+3
```

Op het scherm verschijnt nu tevens de uitdrukking PRINT 2+3. Pas als we op de RETURN-toets drukken, zal de computer op deze opdracht reageren. Deze RETURN-toets wordt zowel met het woord RETURN als met het symbool ↵ aangegeven.

PRINT wil zeggen 'druk af' of liever 'beeld af' en direct na het indrukken van de RETURN-toets zien we ook dat iets wordt afgebeeld, namelijk:

```
5
```

en de term:

```
Ok
```

Het zal duidelijk zijn, dat 5 het antwoord is van de aangegeven optelling 2+3. De term Ok geeft aan dat de computer klaar is met zijn taak en dat we dus weer een nieuwe taak kunnen opgeven.

Opmerkingen:

- Bij de computer mogen we absoluut niet intypen 2+3= (en daarna bijvoorbeeld op de RETURN-toets drukken). Als we dit doen, verschijnt een mededeling – uiteraard in het Engels – dat we iets fout hebben gedaan.
- Bij de computer moet een opdracht, waarmee we onmiddellijk een resultaat op het scherm willen zien, altijd met de term PRINT beginnen.

Vermenigvuldigen en delen

Voor het vermenigvuldigen wordt het teken * gebruikt en voor het delen de schuine deelstreep. Zo geeft PRINT 5*36 het getal 180 als resultaat en PRINT 180/36 de waarde 5.

Haakjes Haakjes mogen we toepassen zoals we dit op de schoolbanken hebben geleerd; zo wordt

$$\frac{5+15}{23\,(17+103)}$$

als volgt ingetoetst:

```
PRINT (5+15)/(23*(17+103))
```

Merk allereerst op dat de totale uitdrukking hier op één regel is geplaatst. Teller en noemer zijn nu tussen haakjes geplaatst. Als we dat niet hadden gedaan, zou de PRINT-instructie er als volgt uitzien:

```
PRINT 5+15/23*(17+103)
```

Deze instructie leidt echter tot een ander dan het gewenste resultaat (dit wordt pas duidelijk als we de zo dadelijk te bespreken prioriteitsregels voor berekeningen kennen).

Bovendien zien we dat in de noemer tussen 23 en 'haakjes openen' een vermenigvuldigteken is geplaatst, het gaat bij de oorspronkelijke uitdrukking van de noemer '23(17+103)' immers om een vermenigvuldiging van 23 met (17+103).

Volgorde waarin de bewerkingen worden uitgevoerd Op de schoolbanken leerden we regels zoals 'Meneer Van Dalen Wacht Op Antwoord'. Daarmee werd aangegeven dat bijvoorbeeld Vermenigvuldigen vóór Delen gaat, of in deftiger taal; vermenigvuldigen heeft een hogere prioriteit dan delen. Bij MSX–BASIC worden de volgende prioriteitsregels in acht genomen:

1. eerst worden uitdrukkingen tussen haakjes uitgewerkt;
2. vermenigvuldigen en delen hebben gelijke prioriteit en gaan vóór optellen en aftrekken die ook gelijke prioriteit hebben;
3. bewerkingen met gelijke prioriteit worden van links naar rechts afgewerkt.

Bijvoorbeeld:

```
PRINT  15/3*5          wordt  5*5          en dus  25
```

Ergo, het resultaat is 25 (en niet 1!)

In Appendix F vindt men een overzicht van de volledige prioriteitsregels.

Tekst De af te beelden tekst plaatsen we steeds tussen aanhalingstekens. Een voorbeeld:

```
PRINT "DIT IS EEN VOORBEELD"
```

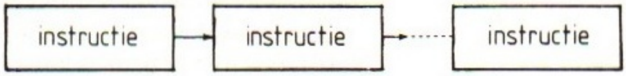
geeft als resultaat:

```
DIT IS EEN VOORBEELD
```

Merk op dat de aanhalingstekens niet bij het resultaat verschijnen. Straks zullen we nog andere mogelijkheden behandelen.

2 EN NU EEN BASIC-PROGRAMMA!

Een programma is niets anders dan een serie instructies die na een zeker start-commando automatisch door de computer wordt afgewerkt.
In schema:



In het meest eenvoudige geval kunnen we een programma, als ging het om een stuk tekst, gewoon op het toetsenbord intypen. Dit houdt in dat zo'n programma, kortom zo'n reeks instructies, eerst in het geheugen wordt opgeslagen.

Hoe krijgen we nu voor elkaar dat de instructies die het programma vormen eerst in het geheugen van de computer worden opgeslagen en niet onmiddellijk worden uitgevoerd? Het antwoord is eenvoudig:

door nummers voor de instructies te plaatsen.

In het nu volgende zullen we dit aan de hand van een voorbeeld illustreren.

Een voorbeeld: Stel dat we de computer als programma achtereenvolgens de volgende 3 instructies willen laten uitvoeren:

```
PRINT "DIT IS"
PRINT "HET EERSTE VOORBEELD"
PRINT "VAN EEN PROGRAMMA"
```

Dan moeten we bij het intypen van dit programma – het gaat tenslotte om een *reeks* instructies – vóór iedere instructie een nummer plaatsen.
We nummeren gewoonlijk met 10, 20, 30 enz.
Dus typen we in:

```
10 PRINT "DIT IS"
20 PRINT "HET EERSTE VOORBEELD"
30 PRINT "VAN EEN PROGRAMMA"
```

Aan het einde van iedere regel drukken we op de RETURN-toets, ten teken dat de regel volledig is ingetoetst (N.B. niet vergeten ook na de laatste regel op de RETURN-toets te drukken!).

Als zo alle regels zijn ingetoetst, is het programma in zijn geheel in het geheugen opgeslagen. We zien het trouwens ook op het scherm staan.

We gaan het programma uitvoeren!

Als we nu intypen:

```
RUN    (gevolgd door RETURN-toets)
```

dan verschijnt op het scherm:

```
DIT IS
HET EERSTE VOORBEELD
VAN EEN PROGRAMMA
```

Hoera, we hebben een programma op de computer verwerkt!

In dit geval bestond het programma uit alleen maar PRINT-instructies met af te drukken tekst, maar niettemin ging het om een reeks instructies, met andere woorden om een programma.

We plaatsen hierbij de volgende opmerkingen:

- Nadat het programma is uitgevoerd – gerund, zeggen computerdeskundigen – blijft het nog steeds in het geheugen staan. Alleen als we een zgn. geheugen-wis-commando geven, dan wel als we de spanning van de computer uitzetten, gaat het programma verloren.
- De nummers geven tevens de volgorde aan waarin de instructies worden afgewerkt en wel van het laagste tot het hoogste getal. We hadden zelfs mogen intypen:

```
10 PRINT "DIT IS"
30 PRINT "VAN EEN PROGRAMMA"
20 PRINT "HET EERSTE VOORBEELD"
```

Het resultaat zal hetzelfde zijn en wel omdat de regelnummers dat zo aangeven. De computer zal de instructies trouwens direct na het intypen in de juiste volgorde plaatsen.

- In plaats van nummers spreken we gewoonlijk van regelnummers. Als regelnummers mogen we alleen gehele getallen (van 0 tot 65529) gebruiken.
- Het feit dat we nu met 10, 20, 30 enz. nummeren, heeft het voordeel dat we naderhand nog instructies kunnen invoegen.
- RUN is een commando d.w.z. de computer moet direct na het intoetsen van dit commando de aangegeven actie uitvoeren.

Andere bekende commando's zijn:

LIST voor de weergave van het programma zelf. Type maar eens LIST gevolgd door de RETURN-toets!

AUTO voor het automatisch genereren van regelnummers.

RENUM voor het hernummeren van regelnummers.

DELETE voor het wissen van gedeeltes van het programma.

NEW voor het wissen van het geheugen.

Deze commando's worden uitvoerig in het Overzicht MSX-BASIC-opdrachten beschreven. Probeer ze één voor één uit!

Het toevoegen, invoegen en wijzigen van regels

In de praktijk komt het vaak voor dat we een programma moeten wijzigen. We illustreren dit aan de hand van het volgende programma:

```
10 PRINT "DIT IS EEN"
20 PRINT "PROGRAMMA"
30 PRINT "TER ILLUSTRATIE"
```

Toets het programma nu in en vergeet niet een eventueel vorig programma met NEW te wissen.

Toevoegen

Toevoegen verloopt min of meer vanzelfsprekend. Kies een regelnummer dat volgt op het laatste regelnummer. Bijvoorbeeld:

```
40 PRINT "VAN HET WERKEN"
50 PRINT "MET REGELNUMMERS"
```

Geef het LIST-commando om u ervan te overtuigen, dat deze regels inderdaad zijn toegevoegd.

Invoegen

Voegt men een regel in dan kiest men een regelnummer dat ligt tussen de regelnummers van de regels waartussen onze nieuwe regel moet komen te staan. Bijvoorbeeld:

```
15 PRINT "KORT EENVOUDIG"
```

Controleer met het LIST-commando of deze regel na te zijn ingetoetst inderdaad op de juiste plaats is gekomen.

Verwijderen

Men kan een regel eenvoudig verwijderen door alleen het betreffende regelnummer in te toetsen gevolgd door de RETURN-toets, bijvoorbeeld:

`15` (RETURN-toets)

Controleer deze actie weer met het LIST-commando. Eventueel kan men ook het DELETE-commando gebruiken (zie het Overzicht MSX-BASIC-opdrachten).

Herstellen van fouten

In het voorafgaande hebben we in feite al enkele mogelijkheden besproken om een programma te wijzigen. In deze paragraaf tonen we nog wat we kunnen doen als er een fout gemaakt is.

De ogenschijnlijk meest eenvoudige oplossing is dan de regel met de fout opnieuw in te typen, maar dan zonder fout.

Voorbeeld

Typ nu in:

```
20 PRINT "KORD EENVOUDIG"
```

Na het LIST-commando zien we hoe deze regel met deze opval-

lende fout in het programma is opgenomen. We kunnen de fout dus herstellen door deze regel opnieuw, maar dan foutloos, in te typen.

Een meer elegante oplossing bestaat uit het werken met de cursor-toetsen. Dit zijn de toetsen met pijltjes rechts van het toetsenbord. Druk maar eens op deze toetsen en we zien hoe het witte blokje zich over het scherm verplaatst.

Dit blokje noemen we de cursor en op grond hiervan worden de toetsen met de pijltjes, de cursorbesturingstoetsen genoemd. Verplaats de cursor nu zodanig dat deze precies op onze fout komt te staan, dus:

```
20 PRINT "KOR[D] EENVOUDIG"
             ↑
```

wijst de cursor aan

Nu drukken we op T en op de RETURN-toets. Vervolgens verplaatsen we onze cursor weer tot onder het programma... en ziedaar de fout is hersteld!

Overigens kunnen we een cursor-toets ook ingedrukt houden, waardoor het effect ontstaat alsof herhaaldelijk op deze toets gedrukt wordt. Dit 'autorepeat-effect' geldt trouwens ook voor de overige toetsen.

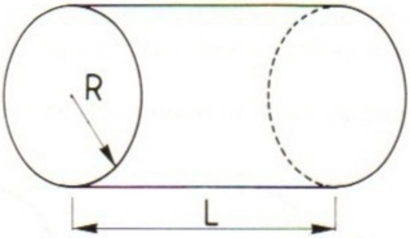
Speciale toetsen

Ten slotte vermelden we nog dat we tijdens het typen onze tekst weer kunnen wissen en wel door op de toets met BS te drukken. Met de INS-toets kunnen we tekens invoegen en met de toets DEL kunnen we tekens wissen. Deze laatste twee toetsen zijn vooral van belang als we met de cursor-toetsen werken. Eerst verplaatsen we de cursor dan naar de plek waar de fout is geconstateerd om vervolgens naar wens tekens te veranderen, te wissen of toe te voegen.

3 VARIABELEN EN NOG MEER OVER REKENEN

Introductie We starten met een inleiding

Om de inhoud van een staaf te bepalen, dienen we de oppervlakte van de doorsnede te kennen en die waarde te vermenigvuldigen met de lengte. In beeld:



oppervlakte van de doorsnede = 3,14159×R×R

inhoud van staaf = oppervlakte van doorsnede × L = 3,14159×R×R×L

We zien hoe de inhoud van een staaf in het algemeen van twee grootheden afhangt; de straal R en de lengte L.

Het volgende programma berekent de inhoud van een staaf en wel voor de waarden:

L=5
en R=7

Programma:

```
10 L=5
20 R=7
30 PI=3.14159
40 OPP=PI*R*R
50 I=OPP*L
60 PRINT I
```

Na het RUN-commando verschijnt:

```
769.68955
```

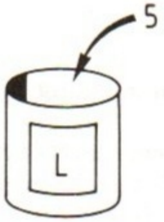
We zullen dit programma nu regel voor regel doornemen. In regel 10 ziet men de uitdrukking;

L=5



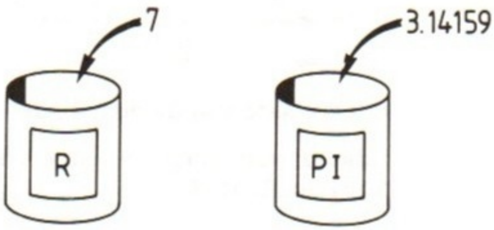
Het effect van deze uitdrukking is dat de computer een gedeelte van het geheugen reserveert en hier als het ware het label L opplakt. Zo'n gedeelte van het geheugen kunnen we het beste als een doos opvatten.

De uitdrukking L=5 wil dan zeggen dat in deze doos het getal 5 wordt opgeslagen. In beeld:



Als in het programma nu naar L verwezen wordt (regel 50), zal de computer in feite naar de *inhoud* van die doos kijken.

Regels 20 en 30 kunnen we evenzo in beeld brengen:



Merk op dat in PI het getal 3.14159 wordt opgeslagen en niet 3,14159. Kortom de komma, die wij gewoonlijk in getallen noteren, wordt in BASIC steeds met een punt aangegeven. Ook bij het resultaat van het programma zien we deze 'decimale punt'.

Als men al enige wiskundige achtergrond heeft, zal men in het getal 3.14159 het getal π (uitspraak: pi) hebben herkend.

Regel 40 toont een bewerking: in de doos met de naam OPP zal de waarde worden opgeslagen die men krijgt door PI met R×R te vermenigvuldigen.

Ergo, in OPP zal de waarde 3.14159 ×7×7 worden opgeslagen.

De zo gevonden waarde wordt weer in regel 50 gebruikt om uiteindelijk de waarde van de inhoud te bepalen. Deze wordt in de doos met het label I opgeslagen.

Ten slotte gebruiken we een PRINT-instructie om de inhoud van deze laatste doos af te beelden:

```
PRINT I
```

↗ ↖

beeld af *de inhoud van de doos met het label I*

Let wel, hier wordt niet de letter I afgedrukt, dan zou de instructie PRINT "I" moeten luiden, met andere woorden dan hadden we I tussen de bekende aanhalingstekens moeten plaatsen.

We plaatsen bij dit eerste programma de volgende opmerkingen:

- We hadden in plaats van

```
10 L=5
```

ook mogen schrijven

```
10 LET L=5
```

Letterlijk staat er dan 'Laat L gelijk aan 5 zijn' of nog beter 'L *wordt* 5'.

Met de laatste uitdrukking benadrukken we dat de computer een *actie* uitvoert: er *wordt* iets in de doos met het label L gestopt. Deze actie wordt helemaal duidelijk bij het volgende voorbeeld:

```
10 LET X=3
20 LET X=X+4
```

Bij de tweede regel wordt in X de oude waarde van X, met andere woorden 3+4 gestopt. Zo zal X uiteindelijk de waarde 7 bevatten, let wel, een uitdrukking als X=X+4 is in BASIC volledig correct, terwijl zij in de wiskunde niet is toegestaan.

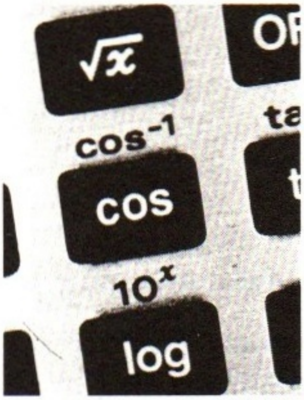
- Het feit dat we een willekeurige waarde in zo'n doos kunnen stoppen, geeft aan dat de inhoud variabel is. Dit is dan ook de reden dat we doorgaans van een *variabele* in plaats van een doos spreken. Zo kent ons programma de variabelen L, R, PI, OPP en I.
In het vervolg zullen we dan ook spreken van 'het toekennen van waarden aan een variabele' in plaats van 'het opslaan van een waarde in een doos'.

- Voor de naamgeving van variabelen gelden bepaalde regels.

Deze zijn als volgt:
- Alleen de eerste twee letters van een naam worden door de computer als *de* naam herkend. Zo zijn de namen EERSTE en EEN voor de computer gelijk: alleen EE wordt herkend.
- We mogen geen gereserveerde woorden als naam gebruiken, zo mag men bijvoorbeeld IF niet als naam gebruiken, omdat deze term gereserveerd is.

Standaardfuncties

Het getoonde voorbeeld was nog zeer eenvoudig. We kunnen ons nauwelijks voorstellen dat we op deze manier ingewikkelde berekeningen kunnen uitvoeren. Met de behandeling van de zgn. standaardfuncties wordt het inzicht met betrekking tot wat we met BASIC zoal kunnen berekenen aanmerkelijk vergroot.



Om de standaardfuncties uit te leggen, maken we eerst weer een uitstapje naar de calculator en wel naar de calculator met wiskundige functies. Als we op zo'n calculator de wortel uit een getal moeten berekenen, voeren we een getal in en drukken daarna op de toets waar het wortel-teken op staat. Zo berekent men, door eenvoudig de daartoe aanwezige toets in te drukken, de gewenste functie.

Bij BASIC kunnen we even eenvoudig waarden voor functies berekenen. Zo'n functie roepen we dan aan met een bepaalde term, en tussen haakjes plaatsen we het getal waarvan de functiewaarde berekend moet worden, dus;

term (waarde)

↗ ↖

geeft functie aan — *geeft waarde aan, waarvan functie bepaald moet worden*

Voorbeeld:

```
10 A=SQR(4)
20 PRINT A
```

Resultaat:

```
2
```

In regel 10 wordt aan A de wortel uit 4 toegekend. De term, die in dit voorbeeld de functie aanduidt, is gelijk aan SQR. Tussen haakjes staat de waarde waarvan de wortel bepaald moet worden, in dit geval (voor de eenvoud) 4.

We plaatsen hierbij nog de opmerking dat tussen de haakjes ook een variabele mag staan, of zelfs een rekenkundige uitdrukking.

Voorbeelden:

`B=SQR(A)` :	hier wordt de waarde door middel van de variabele A aangegeven.
`C=SQR(SQR(5)+4+A)`:	hier wordt de waarde door middel van de SQR(5)+4+A aangegeven. Merk op dat deze uitdrukking zelf ook weer een functie bevat.

In de volgende tabel worden de bekendste functies getoond.

Het Overzicht MSX-BASIC-opdrachten geeft een overzicht van alle functies.

BASIC-aanduiding	*Algebraïsche notatie*	*Betekenis*
ABS (X)	$\|x\|$	Geeft de absolute waarde van X. Dit is de waarde 'zonder teken'.
ATN (X)	arctan (x)	Geeft de arctangens van X. Uitkomst tussen –pi/2 en pi/2.
COS (X)	cos (x)	Geeft de cosinus van X in radialen.
EXP (X)	e^x	Geeft de x-macht van het grondgetal e.
INT (X)	geen	Rondt de waarde van X altijd naar beneden af. Zo geeft INT (3.8) de waarde 3 en INT (–3.1) de waarde –4. We kunnen een getal 'echt' afronden met INT (X+0.5).
LOG (X)	$^e\log (x)$	Geeft de zgn. natuurlijke logaritme van X.
SGN (X)	geen	Geeft de waarde –1, 0, of 1 en wel afhankelijk van het feit of X negatief, 0 of positief is.
SIN (X)	sin (x)	Geeft de sinus van X in radialen.
SQR (X)	$\sqrt{x}$	Geeft de wortel uit X.
TAN (X)	tan (x)	Geeft de tangens van X in radialen.

We merken op dat machtsverheffen in deze tabel niet voorkomt. Hiervoor hanteren we het teken ∧. Zo geeft PRINT 2 ∧ 3 de waarde 8 als resultaat ($=2^3$).

Meer instructies op één regel

BASIC biedt ons de mogelijkheid om meer instructies op één regel te plaatsen.

We dienen het teken: als scheidingsteken te gebruiken. Het volgende programma illustreert dit.

zonder scheidingsteken

```
10 A=10
20 B=5
30 C=A+B
40 PRINT C
```

met scheidingstekens

```
10 A=10:B=5:C=A+B:PRINT C
```

4 INPUT, READ EN DATA

INPUT De eerste instructie die we nu behandelen, is de INPUT-instructie. Deze biedt ons de mogelijkheid om, tijdens het uitvoeren van een programma, waarden aan variabelen toe te kennen. Wellicht nog belangrijker is het feit dat deze instructie ons de mogelijkheid biedt programma's voor algemeen gebruik op te stellen. We bekijken allereerst nogmaals het programma van het vorige hoofdstuk:

```
10 L=5
20 R=7
30 PI=3.14159
40 OPP=PI*R*R
50 I=OPP*L
60 PRINT I
```

We hadden in plaats van dit programma natuurlijk ook kunnen intoetsen:

```
PRINT 3.14159*7*7*5
```

om eenzelfde resultaat te verkrijgen.

Dat we dat niet gedaan hebben, is omdat we wilden tonen hoe we een rekenschema in het geheugen van de computer kunnen opslaan, waarbij voor het bepalen van het resultaat, alleen maar waarden van R en L hoeven te worden ingevuld.

Om bijvoorbeeld de inhoud van een staaf met $L=34$ en $R=10$ te bepalen, kunnen we nu regel 10 en 20 wijzigen in:

```
10 L=34
20 R=10
```

en na het RUN-commando verschijnt het gewenste antwoord. Nu is het wijzigen van een programma, voor het uitvoeren van een berekening, ook niet erg aan te bevelen. We zouden immers fouten kunnen maken...

Het programma zou echter veel beter worden als na het RUN-commando op het scherm mededelingen zouden verschijnen die aangeven dat respectievelijk waarden voor *L* en *R* moeten worden ingevoerd.

Deze mogelijkheid wordt inderdaad geboden en wel in de vorm van de INPUT-instructie. Deze heeft de volgende vorm:

INPUT 'af te beelden tekst'; naam variabele

Bijvoorbeeld:

```
INPUT "VOER LENGTE IN"; L
```

Het resultaat van deze instructie is dat de computer de tekst afbeeldt, inclusief een vraagteken en onmiddellijk daarna het programma onderbreekt. De gebruiker dient nu eerst een getal in te toetsen, bijvoorbeeld:

5   (en RETURN-toets)

Onmiddellijk na het indrukken van de RETURN-toets wordt dit getal dan aan de variabele *L* toegekend en de computer zal weer doorgaan met het uitvoeren van het programma.

Ons programma komt er met INPUT-instructies nu als volgt uit te zien:

```
10 INPUT "VOER LENGTE IN"; L
20 INPUT "VOER STRAAL IN"; R
30 PI=3.14159
40 OPP=PI*R*R
50 I=OPP*L
60 PRINT "DE INHOUD="; I
```

Een voorbeeld van een resultaat is:

```
VOER LENGTE IN? 5
VOER STRAAL IN? 7
DE INHOUD= 769.68955
```

We zien hoe bij de PRINT-instructie in dit programma tevens tekst is opgenomen en wel door deze tekst tussen aanhalingstekens te plaatsen en het teken; als scheidingsteken te gebruiken.

Het programma is ten opzichte van ons eerste programma aanmerkelijk verbeterd. De structuur ligt als het ware volledig vast. We hoeven geen veranderingen in het programma zelf aan te brengen, als we de inhoud van een *willekeurige* staaf willen berekenen.

Opmerkingen: Bij de INPUT-instructie plaatsen we nog de volgende opmerkingen:

- We mogen méér variabelen in een INPUT-instructie opnemen, bijvoorbeeld:

```
INPUT "VOER A, B EN C IN:"; A,B,C
```

Na de onderbreking van het programma toont de computer de tekst en een vraagteken en moeten we drie getallen, gescheiden door een komma, invoeren.

- De tekst in de INPUT-instructie mogen we ook weglaten, bijvoorbeeld:

```
INPUT A
```

Na de onderbreking toont de computer nu enkel nog het vraagteken ten teken dat we een getal moeten invoeren.

READ en DATA De derde, nu te bespreken mogelijkheid voor het toekennen van waarden haakt in op de wens om aan een relatief grote reeks variabelen waarden toe te kennen. Hierbij geldt dan dat deze

waarden doorgaans niet van programma tot programma wijzigingen zullen ondergaan.

We zouden in zo'n situatie natuurlijk ook een hele reeks LET-instructies kunnen opnemen, maar de mogelijkheid met READ en DATA is; zoals we zullen zien, aanzienlijk fraaier. We tonen onmiddellijk een voorbeeld:

```
10 READ A,B,C,D
20 F=A+B+C+D
30 PRINT F
40 DATA 3,7,4,8
```

Het gevolg van de READ-instructie op regel 10 is, dat de computer 'weet' dat er een aantal variabelen genoemd worden, waarvan de waarden worden opgesomd in de regel die met DATA begint.

Zo zal aan A de waarde 3, aan B de waarde 7, aan C de waarde 4 en aan D de waarde 8 worden toegekend.
Het resultaat van het programma bevestigt dit:

```
22
```

Merk op dat de DATA-regel op zich geen uitvoerbare instructie is. Het is gewoon een kladpapiertje, waarop de computer zijn gezochte waarden vindt. We hadden de READ-lijst overigens ook mogen splitsen:

```
10 READ A,B
15 READ C,D
```

en evenzo hadden we de DATA-lijst mogen splitsen, bijvoorbeeld als volgt:

```
40 DATA 3,7,4
50 DATA 8
```

Voor de computer maakt dat alles geen verschil; hij leest de gegevens, als vormen ze één continue lijst. Men kan het mechanisme het beste voorstellen, als plaatst de computer vooraf een pijl bij de eerste waarde van de DATA-lijst. Iedere keer als er nu door middel van een READ-instructie een waarde aan een variabele wordt toegekend, schuift deze pijl één plaats op:

```
40 DATA 3,7,4,8
          ↑
```

pijl schuift steeds één plaats op.

Als de laatste waarde van de DATA-lijst is bereikt, schuift de computer de pijl op naar de eerste waarde van een daaropvolgende DATA-lijst, tenminste als die er is.

We kunnen deze pijl ook terugzetten naar de oorspronkelijke positie en wel met behulp van de volgende instructie:

```
RESTORE
```

Voorbeeld:

```
10 READ A,B
20 RESTORE
30 READ C,D
40 F=A+B+C+D
50 PRINT F
60 DATA 3,7,4,8
```

Resultaat:

```
20
```

Merk op dat de uitkomst nu 20 is. Dit is het gevolg van de RESTORE-instructie; zowel aan A en B als aan C en D wordt nu 3 en 7 toegekend.

5 GETALLEN, GROOT EN KLEIN EN IN ALLERLEI GEDAANTEN

Inleiding Voor het nu volgende gedeelte bekijken we eerst onderstaande tabel.

We kunnen uit de tabel opmaken, dat bijvoorbeeld 10^4 overeenkomt met 10000, met andere woorden het aantal nullen komt hier direct met de macht (cijfer rechtsboven de 10) overeen.

in woorden	*notatie*	*betekenis*	*waarde*
10 tot de macht 1	10^1	10	10
10 tot de macht 2	10^2	10×10	100
10 tot de macht 3	10^3	10×10×10	1000
10 tot de macht 4	10^4	10×10×10×10	10000
10 tot de macht −1	10^{-1}	1/10	0.1
10 tot de macht −2	10^{-2}	1/(10×10)	0.01
10 tot de macht −3	10^{-3}	1/(10×10×10)	0.001
10 tot de macht 0	10^0	10/10	1

Voorbeeld:
Met welke macht komt 1000 000 000 overeen?

Antwoord: tel het aantal nullen, dit zijn er 9 en zodoende vinden we 10^9.

Uit de tabel blijkt tevens dat 0.001 overeenkomt met 10^{-3}. Ook bij getallen kleiner dan 1 kan men kennelijk het aantal nullen tellen om de macht te bepalen.

Voorbeeld:
Met welke macht komt onderstaand getal overeen?

0.000 000 000 000 000 000 1

Antwoord: tel het aantal nullen, dit zijn er 19 en zo vinden we 10^{-19}.

U kunt dit ook schrijven als 1×10^{-19}.

Het eerste getal wordt *mantisse* genoemd en het tweede getal geeft de *exponent* aan.

Bij de computer geven we bovenstaand getal aan volgens de volgende notatie:

```
1E-19
```

Met andere woorden, we gebruiken de letter E om de mantisse van de exponent te onderscheiden.

Om enige ervaring met deze notatie op te doen, kan men het volgende programmaatje gebruiken:

```
10 INPUT A
20 INPUT B
30 C=A*B
40 PRINT C
```

Voorbeelden:

A=987654	B=456789	geeft: 451149483006
A=98765434	B=987654321	geeft: 9.754610765554E+16
A=.0000000008	B=.0000000007	geeft: 5.6E−19
A=8000000000	B=7000000000	geeft: 5.6E+19

Dubbele en enkele precisie

Voor de weergave van getallen gebruikt onze computer een beperkt aantal geheugenregisters (woorden). Dit betekent ook dat berekeningen een beperkte nauwkeurigheid hebben. MSX-BASIC biedt de gebruiker zowel de mogelijkheid om met zgn. dubbele precisie, als om met enkele precisie te werken. Bij dubbele precisie worden steeds 8 registers gebruikt voor de opslag van getallen en bij enkele precisie slechts 4. Standaard werkt onze computer steeds met dubbele-precisie-getallen. Om met enkele-precisie-getallen te werken, plaatsen we steeds een uitroepteken *(!)* achter het getal.

Variabelen waaraan we enkele-precisie-getallen toekennen, onderscheiden zich van dubbele-precisie-variabelen door het plaatsen van *!* direct achter de naam. De volgende voorbeelden illustreren het verschil tussen enkele- en dubbele-precisie-berekeningen.

```
10 A=10/3
20 PRINT A
```

resultaat

```
3.3333333333333
```

```
10 A!=10!/3!
20 PRINT A!
```

resultaat

```
3.33333
```

We plaatsen hierbij nog de volgende opmerkingen:

- Bij dubbele-precisie-getallen mogen we aan het eind van het getal het teken # plaatsen. Evenzo mogen we dit teken als laatste teken van een naam opgeven. Zo geven we expliciet aan, met dubbele-precisie-getallen te werken.
- Als we bij de dubbele precisie-getallen de letter D in plaats van E gebruiken (bijv. 23.45156321D39) dan geven we daarmee expliciet aan dat we met dubbele precisie werken.
- Men dient voortdurend in de gaten te houden of het wel realistisch is, resultaten met zoveel cijfers weer te geven. Stel dat een timmerman een plank van 10 meter in 3 delen moet zagen, dan is volgens onze computer ieder deel 3.3333333333333 m lang... maar kan onze timmerman wel zo nauwkeurig zagen!

- Het gebruik van enkele-precisie-getallen kan vooral van belang zijn als we zuinig met het geheugen willen omspringen. Deze geheugenproblematiek komt in de praktijk vooral naar voren bij de nog te bespreken arrays.

Gehele getallen

In een aantal gevallen willen we in het programma aangeven dat het beslist om gehele getallen, met andere woorden om 'integers' gaat.

Bij de weergave van getallen in een programma levert dat geen problemen op: een getal zonder decimale punt is immers een geheel getal. Aan variabelen kan men echter niet zien of ze een geheel dan wel een getal met een decimale punt voorstellen. Welnu om dat onderscheid wel duidelijk te maken, plaatst men het teken % achter de naam.
Voorbeelden:

```
A%      B1%
```

Let wel, zo'n variabele kan alleen maar een geheel getal bevatten. Als de uitkomst van een berekening een niet geheel getal oplevert, en zo'n waarde wordt aan een integer-variabele toegekend, dan vindt automatisch afronding (naar beneden) plaats.
Voorbeeld:

```
10 A%=20
20 B%=3
30 C%=A%/B%
40 PRINT C%
```

Resultaat:

```
6
```

Merk nogmaals op dat de afronding altijd naar beneden toe plaatsvindt. Bij ons voorbeeld werd 6.6666.. naar 6 afgerond!

Binaire getallen

Om binaire getallen uit te leggen, kijken we eerst naar de opbouw van onze 'gewone', dat wil zeggen decimale getallen. Deze getallen zijn steeds uit machten van 10 opgebouwd (zie tabel aan het begin van dit hoofdstuk).

Voorbeeld:
Het getal 4736 is als volgt uit machten van 10 opgesteld:

4 7 3 6

$6 \times 10^0 = 6 \times 1 = 6$

$3 \times 10^1 = 3 \times 10 = 30$

$7 \times 10^2 = 7 \times 100 = 700$

$4 \times 10^3 = 4 \times 1000 = 4000 +$

4736

We zien hoe ieder cijfer hier met een macht van 10 overeenkomt. Van rechts af tellen we de nulde, eerste, tweede en derde positie en deze posities komen dan precies met de machten van 10 overeen.
Welnu, binaire getallen bestaan alleen uit de cijfers 0 en 1 en de positie van zo'n cijfer komt weer met een macht overeen en wel met een macht van 2.
Voorbeeld:

Met welk decimaal getal komt het binaire getal 10101101 overeen? Uitwerking:

```
1   0   1   0   1   1   0   1
|       |       |   |       |
|       |       |   |       1 ×2⁰=   1
|       |       |   └─────────×2²=   4
|       |       └─────────────×2³=   8
|       └─────────────────────×2⁵=  32
└─────────────────────────────×2⁷= 128
                                   ---
                                   173
```

Bij BASIC kunnen we binaire getallen ook direct aangeven en wel door &B voor het getal te plaatsen.
Voorbeeld:

```
10 A%=&B10101101
20 PRINT A%
```

Resultaat:

```
173
```

Octale en hexadecimale getallen

Na het voorafgaande kunnen we eenvoudig aangeven wat octale en hexadecimale getallen zijn.

Octale getallen zijn getallen waarvan het cijfer overeenkomstig onze uiteenzetting correspondeert met een macht van 8. Bij hexadecimale getallen komt de positie van een cijfer met een macht van 16 overeen.

Het is aardig om op te merken dat we bij decimale getallen precies 10 cijfers hebben (0 t/m 9) om deze getallen te presenteren. Bij binaire getallen zijn er maar twee cijfers (0 en 1) en bij octale zijn er natuurlijk 8 (0 t/m 7).

Tot zover geen problemen. Bij hexadecimale getallen zijn er natuurlijk 16 cijfers... Nu kennen we alleen maar de cijfers 0 t/m 9. Hoe geven we dan de overige cijfers bij hexadecimale getallen aan? Welnu, we gebruiken hiervoor de letters A t/m F.

De volgende tabel toont de decimale getallen 0 t/m 15 en deze worden ter illustratie tevens in binaire, octale en hexadecimale vorm getoond.

decimaal	*binair*	*octaal*	*hexadecimaal*
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Voorlopig lijkt het nut van dit alles nog ver te zoeken. Toch zullen we zien, (onder andere in het hoofdstuk over sprites), hoe deze zaken van pas komen.

Octale en hexadecimale getallen mogen als zodanig direct in programma's worden opgenomen. Voor een octaal getal plaatsen we dan &O en voor een hexadecimaal getal &H.
Voorbeelden:

```
PRINT &O17  geeft  15
PRINT &HF   geeft  15
PRINT &HFF  geeft  255
```

6 PRINT, TAB, LOCATE, PRINT USING EN REM

In vorige hoofdstukken zagen we al dat we meer af te drukken zaken achter een PRINT-instructie mochten plaatsen.

We geven een voorbeeld:

```
10 A=12
20 PRINT "A=";A
30 PRINT "A=",A
```

Resultaat:

```
A=12
A=            12
```

Het verschil tussen het resultaat van regel 20 en dat van regel 30 schuilt in het scheidingsteken. Bij regel 20 gebruikten we de punt-komma als scheidingsteken en bij regel 30 de komma.

Bij een punt-komma komt het volgende af te drukken resultaat direct achter het vorige. Gebruikt men de komma, dan hanteert de computer automatisch een indeling in kolommen.

Bij deze indeling in kolommen wordt steeds gerekend met een tussenruimte van 14 posities.

Overigens mag men aan het einde van een PRINT-instructie deze scheidingstekens ook plaatsen, kijk maar eens naar het volgende voorbeeld:

```
10 PRINT "ABC";"DEFG";
20 PRINT "H";"IJ";"KLM"
```

Resultaat:

```
ABCDEFGHIJKLM
```

Als men bij het afdrukken van tekst een regel wil overslaan, dan gebruikt men een PRINT-instructie zonder nadere aanduiding. Voorbeeld:

```
10 PRINT "A"
20 PRINT
30 PRINT "B"
```

Resultaat:

```
A

B
```

Het is duidelijk dat tussen A en B nu een regel is overgeslagen.

TAB We kunnen aan de hand van de zgn. TAB-functie ook aangeven op welke kolom een af te drukken grootheid moet komen te staan.

Het volgende voorbeeld verduidelijkt het gebruik van TAB.

```
10 PRINT TAB(1);"MSX"
20 PRINT TAB(3);"MSX"
30 PRINT TAB(5);"MSX"
```

Resultaat:

```
MSX
  MSX
    MSX
```

We zien hoe na TAB tussen haakjes steeds de kolom-positie wordt aangegeven vanaf waar de af te drukken zaak (in dit geval tekst) moet komen te staan.

LOCATE De functie TAB regelt de afdrukpositie op de regel. LOCATE geeft zowel de horizontale als de verticale plaats aan. Het is een zeer krachtige instructie, waarmee we op eenvoudige wijze een mooie opmaak voor af te drukken tekst kunnen verkrijgen.



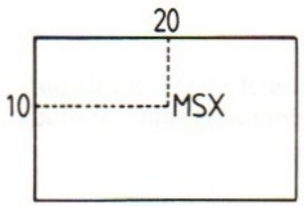
De vorm van de LOCATE-instructie is als volgt:

`LOCATE` horizontale positie , verticale positie

De scherm-indeling is hiernaast aangegeven:

Voorbeeld van een programma:

```
10 CLS
20 LOCATE 20,10
30 PRINT "MSX"
```



geeft het hiernaast afgebeelde resultaat:

Het programma start met een nog niet eerder behandelde instructie, namelijk de CLS-instructie (*Cl*ear-*S*creen-instructie). Deze heeft als resultaat dat het scherm gewist wordt en de cursor in de linkerbovenhoek wordt geplaatst. De daaropvolgende LOCATE-instructie verplaatst de cursor naar de positie (20, 10). Dit houdt dan in dat de tekst MSX vanaf die positie wordt afgebeeld.

WIDTH We merken hierbij nog op dat met de instructie WIDTH het aantal horizontale posities kan worden ingesteld. Zie voor een volledige beschrijving de SCREEN- en WIDTH-instructies in het Overzicht MSX-BASIC-opdrachten.

PRINT USING Bij zeer veel toepassingen willen we dat de resultaten steeds in een bepaalde vorm verschijnen. Als illustratief voorbeeld kunnen we aan financiële berekeningen denken. Uitkomsten dienen hier steeds in een vorm met twee cijfers achter de komma te verschijnen.

De wijze waarop we een getal afbeelden geven we aan met een zgn. PRINT USING-instructie.

Deze heeft steeds de volgende opbouw:

```
PRINT USING  "informatie over representatie";getal
```

We geven een voorbeeld:

```
PRINT USING "##.##"; 32.7
```

Op het scherm verschijnt dan:

```
32.70
```

De uitdrukking tussen de aanhalingstekens, met andere woorden ##.## geeft aan hoe het getal gerepresenteerd moet worden. In dit geval met twee cijfers vóór de decimale punt en twee cijfers áchter de decimale punt.

```
              ##.##
            ↗   ↑   ↖
twee cijfers  de punt  twee cijfers
voor de punt           na de punt
```

De mogelijkheden met betrekking tot PRINT USING zijn bijzonder groot.

Een volledig overzicht wordt in het Overzicht MSX-BASIC-opdrachten gegeven.

REM De laatste instructie die we in dit hoofdstuk bespreken, is de zgn. REM-instructie. De term REM komt van 'remark', hetgeen zoveel betekent als 'opmerking'. We kunnen hiermee commentaar aan een programma toevoegen. Dit commentaar dienen we dan steeds na de term REM te plaatsen.
Een voorbeeld:

```
10 REM ZO MAAR EEN VOORBEELD
20 PRINT "EINDE"
```

Resultaat na het RUN-commando:

```
EINDE
```

We zien dat het na REM geplaatste commentaar na het RUN-commando niet wordt afgedrukt. We zien het alleen maar terug als we weer een LIST-commando geven.

7 DE BESTURINGSINSTRUCTIES: GOTO, IF...THEN, FOR...NEXT EN ON...GOTO

GOTO De GOTO-instructie heeft een zeer eenvoudige vorm en wel:

`GOTO` regelnummer

Als de computer deze instructie tegenkomt, zal een sprong naar het aangegeven regelnummer worden uitgevoerd, alwaar het programma dan wordt voortgezet. Aangezien zo'n sprong *altijd* wordt uitgevoerd, spreken we hier van een *onvoorwaardelijke* sprong.



Een eenvoudig voorbeeld:

```
10 PRINT "DIT WORDT AFGEDRUKT"
20 GOTO 40
30 PRINT "DIT NIET"
40 PRINT "DIT WORDT OOK AFGEDRUKT"
```

Resultaat:

```
DIT WORDT AFGEDRUKT
DIT WORDT OOK AFGEDRUKT
```

Regel 20 geeft een sprong naar regel 40, zodat regel 30 altijd wordt overgeslagen.

IF...THEN De meest eenvoudige vorm van de IF...THEN-instructie is:

`IF` voorwaarde `THEN` instructie

We zien dat tussen de termen IF en THEN kennelijk een voorwaarde is aangegeven en wel een voorwaarde waarvoor geldt dat *ALS* (IF) aan deze voorwaarde voldaan is de na THEN (*DAN*) aangegeven instructie wordt uitgevoerd. We kunnen de constructie het beste aan de hand van een zeer eenvoudig voorbeeld verduidelijken:

```
10 INPUT "VOER GETAL IN";K
20 IF K=3 THEN PRINT "GETAL WAS DRIE"
30 PRINT "EINDE PROGRAMMA"
```

Het programma start met een INPUT-instructie, die aangeeft een getal in te voeren. Stel we voeren de waarde 3 in; aan K wordt dus de waarde 3 toegekend. In de daaropvolgende regel staat een IF...THEN-instructie.



De voorwaarde luidt:

K=3

of in woorden:

is K gelijk aan 3?

Merk op dat we de uitdrukking hier in vragende vorm hebben gesteld. Wel nu, zo'n uitdrukking kan alleen maar goed of fout zijn, of met betrekking tot het voorbeeld; K is inderdaad gelijk aan 3 of dit is niet het geval.

We spreken dan van het al of niet *waar* zijn van de aangegeven voorwaarde. Of om het nog deftiger te zeggen; de logische uitdrukking is al of niet waar. Welnu, aan K was de waarde 3 toegekend en de conclusie is dan ook dat aan de gestelde voorwaarde voldaan is.

In dit geval zal de computer de instructie na THEN uitvoeren, met als resultaat dat de mededeling 'GETAL WAS DRIE' volgt. Als K ongelijk aan 3 zou zijn, met andere woorden aan de gestelde voorwaarde was niet voldaan, dan zal de instructie na THEN gewoonweg worden overgeslagen. Als de computer de IF...THEN-instructie eenmaal heeft afgewerkt, gaat hij door met de daaropvolgende instructie.

In het nu volgende voorbeeld tonen we een programma waar na THEN een GOTO-instructie voorkomt:

```
10 PRINT "HOEVEEL IS 2+5?"
20 INPUT K
30 IF K=7 THEN GOTO 80
40 PRINT "NEE HOOR"
50 PRINT "HET ANTWOORD OP 2+5"
60 PRINT "IS NATUURLIJK 7"
70 GOTO 90
80 PRINT "DAT IS INDERDAAD HET ANTWOORD"
90 END
```

We zien hier dat na THEN de instructie GOTO 80 is opgenomen. Zo zal, als voor K de waarde 7 wordt ingevuld, een sprong naar regel 80 worden uitgevoerd.

Als voor K een waarde ongelijk aan 7 wordt ingevoerd, zal deze GOTO-instructie worden overgeslagen en komen zodoende de regels 40, 50, 60 en 70 aan bod. Merk op dat regel 70 weer een GOTO-instructie aangeeft. Deze is ook noodzakelijk, omdat anders na de tekst van regel 40, 50 en 60 de tekst van regel 80 zou verschijnen... op z'n minst zou dat wat merkwaardig overkomen.

De laatste instructie in dit programma is de zgn. END-instructie, die aangeeft dat het programma is afgelopen. Deze instructie mag eventueel worden weggelaten. In feite hebben we dit bij de voorafgaande programma's ook steeds gedaan.

We plaatsen bij dit programma nog de volgende aantekeningen:

- In plaats van IF K=7 THEN GOTO 80
  had men ook mogen schrijven

  IF K=7 THEN 80

  of met weglating van THEN

  IF K=7 GOTO 80

- Na THEN hadden we eventueel meer instructies en wel gescheiden door de tekens : mogen plaatsen. Voorwaarde is dan wel dat de hele IF...THEN-uitdrukking op één BASIC-regel past (max. 255 tekens).

- In de IF...THEN-instructie wordt K met 7 vergeleken en wel met het teken =. Zo'n teken in een voorwaarde noemen we een relatie-teken. We mogen behalve het teken = ook de volgende tekens gebruiken:

teken	*betekenis*
<	kleiner dan
>	groter dan
< =	kleiner dan of gelijk aan
= <	idem
> =	groter dan of gelijk aan
= >	idem
<>	ongelijk aan
><	idem

- We mogen logische uitdrukkingen ook combineren. De mogelijkheden hiertoe worden in Appendix F besproken.

IF...THEN ...ELSE BASIC staat ook de IF...THEN-instructie toe, waarbij deze met de term ELSE is uitgebreid. We illustreren deze uitbreiding weer aan de hand van een eenvoudig voorbeeld:

```
10 INPUT K
20 IF K=7 THEN PRINT "K IS 7" ELSE
                PRINT "K IS ONGELIJK AAN 7"
30 END
```

De instructie na THEN wordt alleen uitgevoerd als de aangegeven voorwaarde (K=7?) opgaat, in alle andere gevallen wordt de instructie na ELSE uitgevoerd. Ook hier geldt weer, evenals bij THEN, dat we meer instructies na ELSE mogen aangeven en wel gescheiden door het teken dubbelepunt.

FOR...NEXT De FOR...NEXT-instructie dienen we op te vatten als een bijzonder handig programmeergereedschap, waarmee we kunnen aangeven dat een bepaalde serie instructies een aantal keren (herhaald) moet worden afgewerkt.

Het volgende voorbeeld verduidelijkt dit:

```
10 FOR A=1 TO 5
20 PRINT A; "DIT IS EEN DEMONSTRATIE"
30 NEXT A
```

Resultaat:

```
1 DIT IS EEN DEMONSTRATIE
2 DIT IS EEN DEMONSTRATIE
3 DIT IS EEN DEMONSTRATIE
4 DIT IS EEN DEMONSTRATIE
5 DIT IS EEN DEMONSTRATIE
```

De instructies die herhaald moeten worden uitgevoerd, worden steeds ingesloten tussen de regel die met FOR begint en de regel die met NEXT eindigt, dus in schema:

```
.. FOR  naam variabele=...
.. ...  ⎫
.. ...  ⎬ deze instructies worden herhaald uitgevoerd
.. ...  ⎭
.. NEXT naam variabele
```

In ons geval gaat het maar om één instructie, namelijk de instructie van regel 20. Hoe vaak de instructies worden uitgevoerd, kan men geheel en al bepalen door wat in de regel, die met FOR begint, omschreven is.
Bij

```
FOR A=1 TO 5
```

zal het gedeelte precies 5 maal herhaald worden, waarbij A dan achtereenvolgens de waarde 1, 2, 3, 4 en 5 aanneemt. De variabele die na FOR wordt aangegeven noemen we dan ook heel toepasselijk 'de tel-variabele'.

In dit geval wordt A steeds met 1 verhoogd. We mogen echter ook een stap-grootte opgeven en wel door de regel met FOR met de term STEP uit te breiden:

```
10 FOR A=1 TO 6 STEP 2
20 PRINT A; "DIT IS EEN DEMONSTRATIE"
30 NEXT A
40 PRINT A
```

Resultaat:

```
1 DIT IS EEN DEMONSTRATIE
3 DIT IS EEN DEMONSTRATIE
5 DIT IS EEN DEMONSTRATIE
7
```

We zien hoe de tel-variabele A hier achtereenvolgens de waarde 1, 3 en 5 krijgt en voor deze waarde 'de lus' steeds doorlopen wordt.

Als A gelijk aan 7 is, wordt de aangegeven grens in regel 10 (6)

overschreden en vervolgt de computer het programma met regel 40. Nogmaals laten we de waarde van A afdrukken en zo wordt de waarde 7 afgedrukt.

De moraal die wij uit dit programmaatje kunnen trekken is duidelijk ... pas op als we na een FOR...NEXT-instructie de 'telvariabele' verder in het programma gebruiken; deze heeft niet noodzakelijkerwijze de eindwaarde die in de regel met FOR wordt aangegeven!

Nog wat voorbeelden:

- `FOR A=10 TO 5 STEP-1`
  de reeks instructies wordt nu uitgevoerd voor
  A= 10, 9, 8, 7, 6 en 5
- `FOR A= -15 TO 15 STEP 3`
  de reeks instructies wordt nu uitgevoerd voor
  A=− 15, −12, −9, −6, −3, 0, 3, 6, 9, 12 en 15
- `FOR A= 1.4 TO 1.7 STEP .05`
  de reeks instructies wordt nu uitgevoerd voor
  A= 1.4, 1.45, 1.50, 1.55, 1.60, 1.65 en 1.70
- `FOR A=B TO C STEP K/L`
  We zien hier hoe zowel de begin- en grenswaarde als de stapgrootte door variabelen worden aangegeven. De stapgrootte wordt zelfs door een rekenkundige uitdrukking (deling van 2 variabelen) aangegeven. Het is inderdaad toegestaan om begin- grens- en stapwaarde door dit soort uitdrukkingen aan te geven.
- FOR...NEXT-instructies mogen op zich ook weer FOR...NEXT-instructies omvatten. Voorwaarde is dan wel dat de ene instructie de andere volledig omvat. Zo is de constructie

```
FOR K=1 TO 10
...
FOR J=1 TO 5  }
...           }  binnenlus wordt geheel
...           }  omvat door buitenlus
NEXT J        }
NEXT K
```

wel toegestaan en is de constructie:

```
FOR K=1 TO 10 }
...           }
FOR J=1 TO 5  }  'K-lus'  }
...           }           }
...           }           }  'J-lus'
NEXT K        }           }
...                       }
NEXT J                    }
```

verboden. Hier is de binnen-lus immers niet volledig door de buitenlus omvat.

ON...GOTO De laatste instructie die we in dit hoofdstuk bespreken is de ON...GOTO-instructie. Deze instructie doet aan een keuze-schakelaar denken.

Op grond van de waarde die tussen ON en GOTO staat wordt een bepaalde sprong uitgevoerd.
Een voorbeeld verduidelijkt dit:

```
10 INPUT K
20 ON K GOTO 30,50
30 PRINT "K IS 1"
40 GOTO 70
50 PRINT "K IS 2"
60 GOTO 70
70 END
```

Als K gelijk is aan 1 zal een sprong naar het eerste aangegeven regelnummer volgen. Is K gelijk aan 2 dan volgt een sprong naar het tweede regelnummer.

In dit voorbeeld werden na GOTO (regel 20) maar 2 regelnummers opgegeven, in feite kunnen dit er veel meer zijn. Merk tevens op dat de verschillende programma-gedeeltes waar naartoe gesprongen wordt weer worden afgesloten met een GOTO-instructie. Het gebruik van GOTO-instructies is hier weer onvermijdelijk.

8 DE ARRAY (DE LIJST)

Inleiding

In een aantal gevallen bestaat de behoefte om in een programma over een groot aantal variabelen te kunnen beschikken.

We zouden bijvoorbeeld kunnen denken aan een programma voor een voorraadadministratie. Als we voor ieder artikel dat we beheren in een programma een afzonderlijke variabele moeten introduceren, wordt het programma uiteraard zeer groot. In het nu volgende bespreken we hoe BASIC deze problemen ondervangt en wel door de mogelijkheid te bieden via één instructie een groot aantal variabelen te introduceren.

DIM-instructie

De instructie waarmee men eenvoudig een groot aantal variabelen introduceert, is de DIM-instructie. Deze heeft steeds de volgende vorm:

`DIM` naam (aantal)

bijvoorbeeld:

```
DIM A(100)
```

Met bovenstaande instructie introduceert BASIC een reeks van 101 variabelen. De *naam* van ieder van deze variabelen bestaat dan uit de naam, zoals die in de DIM-instructie voorkomt met een getalsaanduiding tussen haakjes. Zo vormen

```
A(0) A(1) A(2) A(3) ... A(99) A(100)
```

precies de 101 variabelen die behoren bij de instructie DIM A(100). We spreken in zo'n geval ook wel eens van de array A, die bij dit voorbeeld bestaat uit de elementen A(0) t/m A(100).

Al die variabelen kunnen we op gelijke wijze als onze 'gewone' variabelen gebruiken.

Een voorbeeld:

```
10 DIM A(100)
20 A(3)=6
30 A(27)=5
40 A(98)=A(3)+A(27)
50 PRINT A(98)
```

Resultaat:

```
11
```

Dit op zich wat merkwaardig aandoende programma toont dat we array-elementen A(3), A(27) en A(98) gebruikt hebben, als ging het om 'gewone' variabelen met bijvoorbeeld de namen A, B en C.

De mogelijkheden van array-variabelen worden vooral bepaald door het feit dat we het nummer tussen haakjes (vaak indexnummer genoemd) ook indirect mogen aangeven.

Voorbeelden:

A(93) nummer wordt hier direct door een getal aangegeven.

A(K) nummer wordt indirect door een variabele aangegeven.

A(K+3) nummer wordt hier indirect door een uitdrukking aangegeven.

Het volgende programma geeft een eenvoudige illustratie:

```
10 DIM B(20)
20 FOR K=1 TO 20
30 B(K)=K
40 NEXT K
50 FOR K=20 TO 1 STEP -1
60 PRINT B(K);
70 NEXT K
```

Resultaat:
```
20 19 18 17 16 15 14 13 12
11 10 9 8 7 6 5 4 3 2 1
```

In regel 10 wordt array B geïntroduceerd. Regels 20 t/m 40 geven aan dat aan B(1) de waarde 1, B(2) de waarde 2, etc. moet worden toegekend. Dat die waarden inderdaad zijn toegekend, bewijzen de regels 50 t/m 70, als gevolg waarvan de waarden van de variabelen B(20), B(19), B(18), etc. worden afgedrukt. We merken tevens op hoe handig de FOR...NEXT-instructie in combinatie met arrays kan worden gebruikt.

We maken bij de array nog de volgende opmerkingen:

- Als in een programma een array-variabele bijv. A(6) wordt gebruikt, zonder dat in dat programma een DIM-instructie is vermeld, neemt de computer een array met een bovengrens van 10 aan. Bij het voorbeeld gaat de computer er dus van uit dat de variabele A(6) behoort bij array A die als bovengrens 10 heeft (DIM A(10)).

- We mogen ook zogenaamde meer-dimensionale arrays in een programma gebruiken, d.w.z. tussen de haakjes worden nu meer getallen aangegeven. Zo introduceert de instructie:

```
DIM A(3,3)
```

de variabelen:

A(0,0)	A(0,1)	A(0,2)	A(0,3)
A(1,0)	A(1,1)	A(1,2)	A(1,3)
A(2,0)	A(2,1)	A(2,2)	A(2,3)
A(3,0)	A(3,1)	A(3,2)	A(3,3)

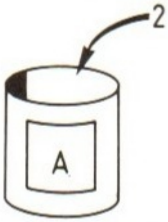
- We mogen op één regel méér arrays introduceren.
Bijvoorbeeld:

```
10 DIM A(100), P(300), Z(50), P!(30), B%(5)
```

- We kunnen de gereserveerde ruimte naderhand indien gewenst, vrijmaken door de instructie ERASE te gebruiken.
Bijvoorbeeld: `ERASE A,P`

9 STRINGS... SPELEN MET LETTERS

Inleiding Tot nu toe hebben we gezien dat we aan een variabele een getal kunnen toekennen. We hadden bij het uitleggen, hoe dit in zijn werk ging, gebruik gemaakt van een voorstelling met een doos. Zo werd de instructie A=2 aanschouwelijk gemaakt volgens de afbeelding hiernaast.



We kunnen aan variabelen ook reeksen letters toekennen. Een dergelijke reeks letters noemen we dan een string en dit is tevens de reden dat we dergelijke variabelen, string-variabelen noemen.

Uiteraard zullen we door een speciale schrijfwijze duidelijk moeten maken dat het om een string-variabele gaat. De afspraak is eenvoudig:

als we onmiddellijk na de naam van een variabele een dollarteken plaatsen, gaat het om een string-variabele.

Zo zijn A$, EERSTE$ en TEKST$ voorbeelden van namen die kennelijk betrekking hebben op string-variabelen.

De tekst die we aan zo'n string-variabele kunnen toekennen, moeten we altijd tussen aanhalingstekens plaatsen.
Het volgende programmaatje toont een voorbeeld:

```
10 LET A$= "DIT IS EEN STRING"
20 PRINT A$
```

Regel 10 kunnen we weer d.m.v. een plaatje verduidelijken.



Merk op dat in regel 10 deze tekst inderdaad tussen aanhalingstekens is geplaatst. Als we nu het resultaat van het programma bekijken, zal opvallen dat deze aanhalingstekens niet worden afgedrukt, als resultaat verschijnt:

```
DIT IS EEN STRING
```

We kunnen dus concluderen dat de aanhalingstekens geen deel uitmaken van de string zelf.

Hieronder volgen nog een aantal voorbeelden van strings die we aan een stringvariabele mogen toekennen.

"HALLO JAN" dit is een string, die bestaat uit negen tekens, de spatie tussen HALLO en JAN telt namelijk ook mee!

"164" dit is een string die bestaat uit drie tekens. Het feit dat het hier om cijfers gaat, is eigenlijk erg verraderlijk want de computer ziet slechts tekens en geen cijfers! Straks zullen we zien hoe we in dit soort bijzondere gevallen, zo'n 'schijngetal' in een echt getal kunnen omzetten.

"" dit is een string die bestaat uit geen enkel teken, de aanhalingstekens zijn namelijk direct na elkaar geplaatst. We zullen straks voorbeelden zien waarom zo'n 'lege-string' toch voordelen kan bieden. Deze string wordt ook wel eens aangeduid met de term nul-string.

Concateneren... of het aan elkaar rijgen van strings

Met behulp van het optelteken + kunnen we aangeven dat twee strings aan elkaar geregen moeten worden. Computerdeskundigen spreken dan van concateneren.

Een voorbeeld verduidelijkt dit:

```
10 A$="MSX-"
20 B$="BASIC"
30 C$=A$+B$
40 PRINT C$
```

Resultaat:

```
MSX-BASIC
```

We zien hoe in regel 10 en 20 de strings 'MSX-' en 'BASIC' aan A$ en B$ worden toegekend. In de daarop volgende regel worden deze strings 'opgeteld' en aan C$ toegekend.

Stringfuncties

Tot nu toe konden we nog niet zoveel met strings doen, hoogstens konden we twee of meer strings aan elkaar rijgen om op deze wijze een langere string te verkrijgen.

Gelukkig kent BASIC een groot aantal functies waarmee we allerlei handelingen met betrekking tot strings kunnen uitvoeren.

We kunnen deze stringfuncties in twee groepen verdelen:

groep 1: de functies die weer een string als resultaat hebben. De namen van deze functies eindigen steeds met $.

groep 2: de functies die als resultaat een getal opleveren. De meeste functies uit deze groep maken gebruik van een bepaalde afspraak. Deze afspraak heeft betrekking op het geven van getallen aan letters en andere tekens. De namen van de functies uit deze groep eindigen niet op een dollar-teken.

Beide groepen functies zijn opvallend eenvoudig van aard. In het Overzicht van MSX-BASIC-opdrachten vindt men een uitgebreide beschrijving en bij iedere functie wordt tevens een voorbeeld gegeven. We volstaan hier met een korte opsomming van de belangrijkste functies.

LEN bepaalt het aantal karakters in een string.

LEFT$ geeft een onderdeel van een string; de zgn. substring (linkergedeelte).

RIGHT$ geeft een substring (rechtergedeelte).

MID$	geeft een substring (algemeen).
ASC	geeft de ASCII-waarde van het eerste karakter.
CHR$	geeft het karakter volgens de opgegeven ASCII-waarde.
VAL	zet een numerieke string in een overeenkomstig getal om.
STR$	zet een getal in een overeenkomstige string om.
SPACE$	voor het afdrukken van een aantal spaties.
INSTR	voor het zoeken van een substring in een gegeven string.

INKEY$ en een spelletje

INKEY$ is in feite geen functie maar een variabele. Als de computer bij het 'runnen' van een programma de term INKEY$ aantreft, 'kijkt' de computer naar het toetsenbord. Als op dat moment een toets wordt ingedrukt, wordt het betreffende teken aan INKEY$ toegekend. Als we op dat moment geen toets indrukken, zal de lege string " " aan INKEY$ worden toegekend.

Het volgende programma illustreert hoe INKEY$ voor een eenvoudig reactiespelletje gebruikt wordt:

```
10 PRINT "DRUK OP EEN TOETS"
20 PRINT "ALS HET SCHERM"
30 PRINT "DE WAARDE 25 TOONT"
40 FOR K=1 TO 50
50 PRINT K
60 A$=INKEY$
70 IF A$ <> "" THEN GOTO 90
80 NEXT K
90 PRINT "EINDE"
```

Nog wat opmerkingen

In het voorafgaande hebben we de belangrijkste functies met betrekking tot strings besproken. Een overzicht van alle toegestane instructies treft men in het Overzicht MSX-BASIC opdrachten aan.

Ten slotte plaatsen we nog de volgende opmerkingen:

- We mogen, evenals dat bij getallen is toegestaan, ook string-arrays introduceren, bijvoorbeeld:

```
DIM A$(100)
```

- De computer heeft bij het aanzetten een bepaalde geheugenruimte gereserveerd voor de opslag van strings. Standaard kunnen zo 200 tekens worden opgeslagen. We kunnen deze ruimte vergroten met een zogenaamde CLEAR-instructie, bijvoorbeeld:

```
CLEAR 500
```

Met deze CLEAR-instructie reserveren we een ruimte voor de opslag van 500 tekens.

- De string die we aan een string-variabele mogen toekennen, heeft een maximale lengte van 255 tekens.

10 SUBROUTINES: GOSUB...RETURN EN DEF FN

GOSUB... RETURN

Een subroutine is een gedeelte van een programma dat vanuit een willekeurige plaats in het programma kan worden aangeroepen. De sprong naar de subroutine vindt steeds plaats aan de hand van de instructie:

`GOSUB` regelnummer

Als de computer deze instructie tegenkomt, zal hij het programma op het aangegeven regelnummer vervolgen. Na het herkennen van de term:

```
RETURN
```

zal de computer weer terugspringen naar het oorspronkelijke programma.

Het nu volgende schema verduidelijkt de situatie.

Hoofdprogramma *Subroutine*

```
10 ...
20 ...                  300...
.. ...                  .. ...
100 GOSUB 300           .. ...
110 ...                 .. RETURN
```

We zien hoe op regel 100 de subroutine wordt aangeroepen. Deze begint op regel 300.

Op het moment dat RETURN wordt aangetroffen, keert de computer terug naar regel 110 d.w.z. vervolgt de computer het oorspronkelijke programma weer. Hiermee is de volledige constructie besproken, waarbij we er nog op wijzen dat in subroutines weer een sprong naar een andere subroutine kan worden opgenomen.

Het volgende programma geeft een illustratie. Met behulp van dit programma kunnen we het bekende spelletje '13 lucifers' spelen. Om beurten mogen we 1, 2 of 3 lucifers van de stapel pakken. Wie de laatste lucifer moet pakken heeft verloren.

```
 10 PRINT "U MOET BEGINNEN"
 20 L=13
 30 GOSUB 80
 40 GOSUB 80
 50 GOSUB 80
 60 PRINT "HA HA, IK HEB GEWONNEN"
 70 END
 80 REM START SUBROUTINE
 90 INPUTK: IFK<0 ORK>3 THEN GOTO 90 ELSE A=4-K
100 PRINT "IK NEEM ER...";A
110 L=L-K-A
120 PRINT "AANTAL OVERGEBLEVEN LUCIFERS=";L
130 PRINT "NU BENT U WEER"
140 RETURN
```

Merk op hoe door middel van END de subroutine van het hoofdprogramma is afgescheiden. Op deze wijze vermijden we dat de subroutine zou worden betreden zonder de instructie GOSUB. Voor alle duidelijkheid: subroutines mogen alleen worden betreden door middel van de GOSUB-instructie.

DEF FN Met behulp van de DEF FN-instructie kunnen we zelf een functie definiëren.

De DEF FN-instructie heeft steeds de volgende vorm:

DEF FN naam (reeks variabelen gescheiden door komma's) = uitdrukking waarin deze variabelen voorkomen

Bijvoorbeeld:

```
DEF FNA (X,Y)=X^2+Y^2
```

De functienaam bestaat uit één letter (in ons voorbeeld de letter A). Bij dit voorbeeld wordt de functie A gedefinieerd door:

X^2+Y^2

Deze functie wordt dan opgeroepen met de uitdrukking FNA, met andere woorden met de term FN gevolgd door de naam van de functie.
Het volgende programma illustreert dit:

```
10 DEF FNA(X,Y)=X^2+Y^2
20 A=3
30 B=4
40 C=FNA(A,B)
50 PRINT C
```

Resultaat:

```
25
```

UITBREIDINGEN VOOR MSX2-BASIC

1

GELUID: BEEP, PLAY EN SOUND

BEEP BEEP wil zo veel zeggen als 'even piepen'.

Onderstaand programma illustreert deze instructie:

```
10 PRINT "BEGIN"
20 FOR K=1 TO 1000
30 PRINT K
40 NEXT K
50 BEEP
60 PRINT "KLAAR"
```

We zien hoe de BEEP-instructie hier aan het eind van het programma is geplaatst. Als de computer klaar is met zijn rekenwerk hoort men een korte pieptoon.

PLAY De PLAY-instructie is geheel en al gericht op het eenvoudig aangeven van melodieën.

Het is een buitengewoon krachtige instructie waarin we o.a. de volgende zaken kunnen aangeven:

- het tempo waarin gespeeld moet worden.
- de octaaf waar een noot betrekking op heeft.
- de duur van een noot.
- de noot (toon) zelf.
- rustmaten.
- het volume waarmee een noot wordt gespeeld.
- bepaalde geluidseffecten.

PLAY kan als instructie, maar ook als commando worden gebruikt.
Toets maar eens in:

```
PLAY "CDE"
```



Na het indrukken van de RETURN-toets horen we:

Kortom de noten C, D en E en wel die behoren bij de octaaf met de G-sleutel, we zullen deze octaaf voortaan aangeven met 'octaaf 4'.

Het volgende programmaatje maakt van de PLAY-instructie gebruik om een eenvoudig orgeltje van onze MSX-computer te maken.

```
10 A$=INKEY$
20 PLAY A$
30 GOTO 10
```

In regel 10 wordt aan A$ de ingedrukte toets toegekend, die dan vervolgens in de PLAY-instructie wordt gebruikt etc.

Ons orgeltje is zeker nog niet perfect, want we kunnen o.a. het tempo in het geheel niet regelen. Gelukkig zijn er veel mogelijkheden om een en ander wel te regelen.

Meer geluidskanalen

Als we drie strings, gescheiden door komma's, na PLAY opnemen, zal de MSX-computer iedere string voor een afzonderlijk geluidskanaal gebruiken. Bijvoorbeeld: bij

```
PLAY "CDE"
```

zal er steeds van één kanaal gebruik worden gemaakt en bij

```
PLAY "CDE", "EFG"
```

zullen tegelijkertijd twee kanalen gebruikt worden en bij

```
PLAY "CDE", "EFG", "GAC"
```

zullen de drie kanalen tegelijkertijd worden gebruikt. Dit houdt in dat de noten C, E en G en hierna D, F en A en ten slotte E, G en C tegelijkertijd ten gehore worden gebracht.

Tempo

Het tempo geven we met een zgn. T-code aan.
Geef maar eens het volgende commando:

```
PLAY "T200CDE"
```

De noten C, D en E worden nu in een hoger tempo dan voorheen gespeeld. Dit klopt, want zonder nadere mededeling neemt de computer 'de stand' T120 aan en T200 geeft zodoende een hoger tempo aan. De T-waarde kan liggen tussen 32 en 256.

Volume

Het volume geven we met een V-code aan.
Ter illustratie:

```
PLAY "T200V13CDE"
```

We horen nu dezelfde wijs als bij het vorige voorbeeld alleen de volume-knop (V) staat kennelijk in een hogere stand. Ook dit klopt want zonder nadere aanduiding neemt de computer 'de

stand' V8 aan en V13 komt zodoende op een groter volume neer. De V-code kan variëren tussen 0 (minimaal) en 15 (maximaal).

De noten en de octaaf

De noten geven we aan met de letters C, D, E, F, G, A en B. Bij halve tonen, bijvoorbeeld de CIS, gebruiken we het teken # (of +). Bijvoorbeeld:

```
PLAY "CC#" en evenzo PLAY "CC+"
```

brengt achtereenvolgens de C en de CIS ten gehore. De vraag is nu 'hoe geven we de hoge C aan?' Welnu, de hoge C behoort bij het volgende octaaf. Nu hebben we in het voorafgaande al vermeld dat zonder nadere mededeling, de computer octaaf 4 (O4) veronderstelt.

Om nu de hoge C te produceren, moeten we dus O5 aangeven.
Bijvoorbeeld:

```
PLAY "CDEO5CDE"
```

brengt tweemaal de reeks CDE ten gehore, maar bij de laatste reeks wordt op een octaaf hoger gespeeld.

De octaaf-code (O-code) kan minimaal 1 en maximaal 8 zijn en hieruit volgt dan dat onze MSX-computer muziek over acht octaven kan spelen.

Duur van de noten

Tot nu toe klonken alle noten even lang. Wie echter een muziekboek openslaat ziet dat tonen een verschillende duur kunnen hebben.

In muziekschrift geven we dat aan door het bolletje al dan niet op te vullen en eveneens door vlaggetjes aan de stok te tekenen.

teken	𝅝	𝅗𝅥	𝅘𝅥	𝅘𝅥𝅮	𝅘𝅥𝅯	𝅘𝅥𝅰	𝅘𝅥𝅱
aantal tellen	4	2	1	1/2	1/4	1/8	1/16
L-code	1	2	4	8	16	32	64

De duur van een noot geven we met de L-code aan. Zonder nadere opgave neemt de computer L4 aan en dit komt dan neer op een kwart noot.

In het volgende illustreren we de L-code aan de hand van 'boer, wat zeg je van m'n kippen'.
Omgezet in een PLAY-commando wordt dit:



```
PLAY "L2GL4EL2GL4EDEFL2EL4C"
```

We mogen dit ook korter opschrijven en wel door onmiddellijk achter een noot de duur aan te geven, dus voor ons voorbeeld:

```
PLAY "G2EG2EDEFE2C"
```

Rusttekens

In muziekschrift komen we ook rusttekens tegen. Deze geven aan dat een bepaalde periode, of liever een bepaald aantal tellen, geen toon gespeeld mag worden. We gebruiken hiervoor de R-code.

Het volgende overzicht toont de tekens van het muziekschrift die hiervoor gebruikt worden en wel samen met de R-codes.

teken 

aantal tellen	4	2	1	1/2	1/4	1/8	1/16
R-code	1	2	4	8	16	32	64

Het volgende voorbeeld geeft een illustratie:

```
10 PLAY "CDER4DECDECR2": GOTO 10
```

Na de eerste E is er een rust van één tel en na de laatste C een rust van twee tellen. Door PLAY op deze wijze in een programma op te nemen, kan men de maat van de muziek nog beter bestuderen.

We onderbreken de 'schoone wijs' met CTRL/STOP. In deze paragraaf over rusttekens merken we ten slotte nog op dat we achter een noot een punt mogen plaatsen. Deze noot zal dan anderhalf maal zolang klinken, bijvoorbeeld:

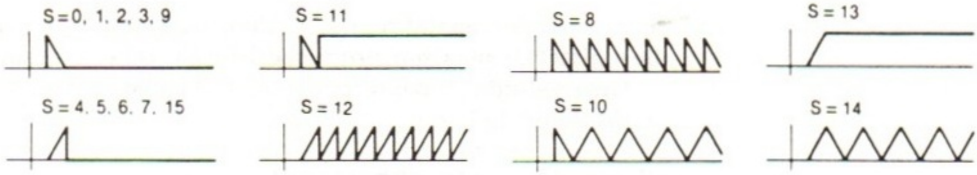
```
PLAY "CD.E"
```

Ook hier kunnen we dus het ritme van onze muziek mee bepalen.

De S- en de M-code

De laatste codes die we bij de PLAY-instructie mogen gebruiken, zijn de S- en de M-code. De S-code heeft te maken met een bepaalde klankkleur die we mee kunnen geven en de M-code bepaalt in hoeverre het patroon van die klankkleur herhaald moet worden.

De S-code geeft als het ware aan, hoe ons geluid volgens een bepaald patroon versterkt en verzwakt moet worden. Hierbij gebruiken we de volgende plaatjes:



De M-code geeft dan de lengte van de cyclus weer, waarin onze met S gekozen vorm herhaald moet worden. Zonder nadere aanduiding neemt de computer voor S de waarde 1 en voor M de waarde 255 aan (M kan liggen tussen 1 en 65535).
Voorbeelden:

```
PLAY "S10M510CDE"
```

en

```
PLAY "S8M6000CDE"
```

We horen bij het eerste voorbeeld bij iedere toon een opvallend stotende klank. Bij het tweede voorbeeld lijkt het wel of de tonen op een piano worden gespeeld. Probeer nu zelf verschillende combinaties uit om de verschillende mogelijkheden van de S- en M-codes te ontdekken.

Een uitgewerkt voorbeeld

Het volgende voorbeeld laat de computer de canon 'vader Jacob' spelen. Men kan zien hoe de strings, die in de PLAY-instructie worden gebruikt, van te voren met string-variabelen worden opgebouwd.

```
 10 CLEAR 500
 20 A$="L4CDECR64CDEC"
 30 B$="EFG2EFG2"
 40 C$="L8GAGFL4ECL8GAGFL4EC"
 50 D$="CO3GO4C2C4O3GO4C2"
 60 W$=A$+B$+C$+D$
 70 W1$=W$
 80 W2$="R1R1"+W$
 90 W3$="R1R1"+W2$
100 PLAY W1$,W2$,W3$
```

(N.B. in de string in regel 50 worden hoofdletters O gebruikt.)

De CLEAR-instructie wordt gebruikt om voldoende ruimte voor de opslag van strings te reserveren. In regel 20, 30, 40 en 50 wordt de melodie in de kenmerkende stukken aangegeven. Regel 60 geeft de combinatie van de strings en zo de hele wijs weer. Regel 80 en 90 bepalen de tweede en derde stem.

SOUND

De laatste instructie die we in dit hoofdstuk bespreken, is de SOUND-instructie. Deze instructie kan men alleen goed begrijpen door te beseffen dat onze afzonderlijke geluids-chip is uit-

gerust met een aantal registers. Door nu bepaalde waarden in die registers te plaatsen, produceert die chip allerlei geluiden.

Onze geluids-chip is uitgerust met 13 registers die de volgende betekenis hebben:

Register-nummer	*wordt gebruikt voor*
0,1	de combinatie van de inhoud van register 0 en 1 bepaalt de frequentie van de toon voor geluidskanaal A.
2,3	evenzo, maar nu voor kanaal B.
4,5	evenzo, maar nu voor kanaal C.
6	bepaalt de klankkleur van een voort te brengen ruisachtig geluid. De waarde kan van 0 tot 31 variëren.
7	bepaalt of een bepaald geluidskanaal ruis, dan wel een toon weergeeft.
8	bepaalt volume van kanaal A (waarde tussen 0 en 15).
9	als 8, maar nu voor kanaal B.
10	als 8, maar nu voor kanaal C.
11, 12	bepaalt de klankkleur en wel de zgn. modulatie van een toon.
13	bepaalt de klankkleur (aanzwel- en uitsterfpatronen).

Om de frequentie van een bepaalde toon te berekenen, maken we gebruik van het volgende hulpprogramma:

```
10 INPUT "GEWENSTE FREQUENTIE"; F
20 A=1789772.5
30 B=INT(A/(16*F))
40 C=INT(B/256)
50 D=B-C
60 PRINT D,C
```

De waarden van D en C zijn dan de waarden die in het eerste en tweede register geplaatst moeten worden om de desbetreffende toon te verkrijgen.

Stel, we willen de A produceren (frequentie 440). Ons programmaatje produceert de waarden:

254 0

Dit leidt dan tot de volgende SOUND-instructies:

```
10 SOUND 0,254
20 SOUND 1,0
30 SOUND 8,11
```

We horen nu de toon A (probeer ook PLAY 'A'). Het geluid kunnen we onderbreken met CTRL/STOP.

Ruis In de praktijk zullen we de SOUND-instructie gebruiken om allerlei opvallende geluidseffecten, zoals gillende sirenes en plofgeluiden te maken. De volgende voorbeelden geven enkele mogelijkheden aan:



```
10 REM FLUITKETEL
20 FOR K=255 TO 0 STEP-1
30 PRINT K
40 SOUND 0,K
50 SOUND 1,0
60 SOUND 8,10
70 NEXT K
```

Laat men regel 30 weg dan verkrijgt men een versnelde variatie van het geluid.

Met het volgende voorbeeld kunnen we nog verschillende effecten uitproberen:

```
10 INPUT K,L,M
20 SOUND 0,250:SOUND 1,0
30 SOUND 6,K:SOUND 7,L:SOUND 13,M
40 FOR J=15 TO 0 STEP -0.05
50 SOUND 8,J
60 NEXT J
```

Voor K=10, L=10 en M=10 levert dit een uitstervende toon, maar voor K=20, L=20 en M=20 hoort men een explosie.



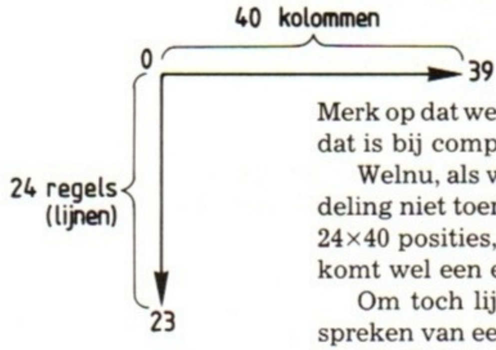
2 GRAFISCHE VOORSTELLINGEN: SCREEN, PSET, COLOR EN PRESET

Inleiding Een grafische voorstelling is niets anders dan een deftig woord voor een plaatje. Plaatjes bouwen we op met punten, lijnen en krommen.

Onze MSX-computer heeft bijzonder veel mogelijkheden om allerlei fraaie figuren op het scherm te tonen. Dat is belangrijk want veel toepassingen zijn juist gebonden aan het werken met boeiende plaatjes.

SCREEN Om inzicht te krijgen hoe we met een MSX-computer plaatjes opbouwen, moeten we eerst iets meer over de indeling van ons scherm (Engels: screen) vertellen.

Bij al de tot nu toe geboden voorstellingen hanteerde de computer een indeling in 24 regels, waarbij iedere regel 40 tekens kan bevatten (na het commando WIDTH 40).



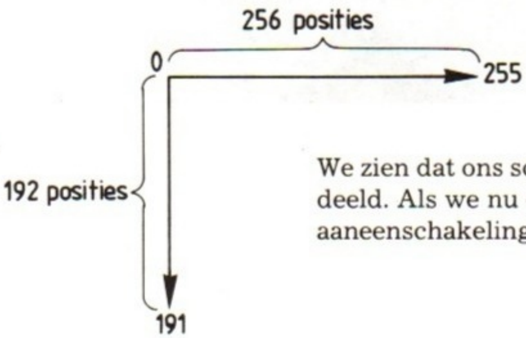
Merk op dat we ons assenstelsel min of meer op z'n kop tekenen: dat is bij computers gebruikelijk.

Welnu, als we fijne tekeningen willen weergeven, is deze indeling niet toereikend. Als we bijvoorbeeld op een scherm van 24×40 posities, door middel van kruisjes een lijn tekenen, dan komt wel een erg grove figuur naar voren.

Om toch lijnen met 'hoge fijnheid' weer te geven (folders spreken van een hoge resolutie), moeten we de computer eerst omschakelen, zodat een andere schermindeling wordt gehanteerd. Na de instructie:

```
SCREEN 2
```

hanteert de computer een indeling die hieronder is aangegeven:



We zien dat ons scherm nu in een veel fijner raster is onderverdeeld. Als we nu een deel van het raster bekijken, zien we een aaneenschakeling van vierkantjes.

Ieder vierkantje noemen we voortaan een beeldpunt (of kortweg punt) en de coördinaten van ieder punt geven we dan door een X- en een Y-waarde aan en wel volgens de volgende afspraak, dat de horizontale as de X-as is en de vertikale as de Y-as.
Hierbij mag X van 0 tot 255 variëren en Y van 0 tot 191.

Punt (0,0) komt dus neer op het punt linksboven en (191,255) met het punt rechtsonder.

Als we met behulp van de instructie SCREEN 2 de computer eenmaal in de toestand hebben gebracht dat het fijne raster gebruikt wordt, kunnen we stippen en lijnen met daartoe beschikbare functies aangeven.

PSET

We starten met de instructie PSET (point*set*, dat wil zeggen, geef een punt weer) waarmee we stippen op het scherm kunnen plaatsen.
In de eenvoudigste vorm ziet deze instructie er als volgt uit:

PSET (x-coördinaat, y-coördinaat)

We starten maar meteen met een voorbeeld:

```
10 INPUT "X=";X
20 INPUT "Y=";Y
30 SCREEN 2
40 PSET (X,Y)
50 GOTO 50
```

Regel 10 en 20 kennen een waarde aan X en Y toe. In regel 30 staat de noodzakelijke instructie SCREEN 2. Daarna wordt met de PSET-instructie de stip getekend. De laatste instructie zorgt er voor dat het programma niet wordt beëindigd (na het beëindigen van het programma wordt automatisch SCREEN 0 uitgevoerd).

We kunnen het programma dan ook alleen maar onderbreken door met ingedrukte CTRL-toets op STOP te drukken (CTRL/STOP).

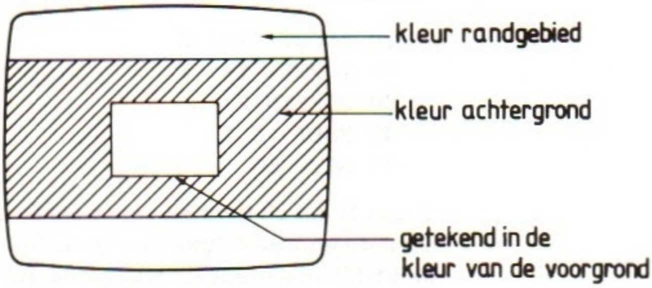
Onze computer is trouwens nu in staat om beelden met een nog grotere fijnheid weer te geven. We dienen dan weer een andere SCREEN-modus aan te geven.De tabel op de volgende pagina toont alle mogelijkheden.

Ogenschijnlijk is er geen verschil tussen een aantal modussen. Zo geven SCREEN 6 en SCREEN 7 beide een indeling in 512×212 pixels. Het verschil is echter dat in de ene modus andere mogelijkheden m.b.t. het toepassen van kleur aanwezig zijn dan in de andere modus.

SCREEN-modus	*Omschrijving*
0	Voor weergave van tekst: 40×24. Met WIDTH 80-commando: 80×24.
1	Voor weergave van tekst: 32×24.
2	Grafisch: 256×192 pixels (=beeldpunten).
3	Grafisch: 64×48 blokken.
4	Grafisch: 256×192 pixels.
5	Grafisch: 256×212 pixels.
6	Grafisch: 512×212 pixels.
7	Grafisch: 512×212 pixels.
8	Grafisch: 256×212 pixels.

COLOR Alvorens we aangeven hoe we kleuren (Amerikaans: colors) kunnen gebruiken, vertellen we eerst iets meer over de termen 'voorgrond', 'achtergrond' en 'randgebied'.

De volgende afbeelding toont de indeling van het scherm.



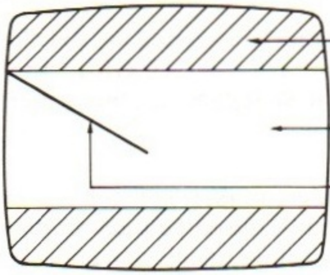
Onder de voorgrond verstaan we tekens, figuren, symbolen enz., die we kunnen afbeelden. Deze zaken kunnen we in een bepaalde kleur afbeelden en we spreken dan van de 'kleur van de voorgrond'.

De kleur waartegen we deze tekens, figuren, symbooltjes enz. afbeelden noemen we de kleur van de achtergrond. Ten slotte zien we dat er nog een randgebied te onderscheiden valt, waarvan we ook nog de kleur kunnen aangeven.

Een voorbeeld

```
10 SCREEN 2
20 COLOR 1,9,12
30 CLS
40 FOR K=1 TO 100
50 PSET (K,K)
60 NEXT
70 GOTO 70
```

Nu verschijnt de volgende afbeelding:



Door de instructies van regel 40 t/m 60 worden 100 punten afgebeeld, die tezamen de zwarte lijn vormen.

In dit voorbeeld komt een COLOR-instructie voor. Deze heeft de volgende vorm:

`COLOR` kleur voorgrond, kleur achtergrond, kleur randgebied

Voor de standaardkleuren dienen we de volgende tabel te hanteren:

cijfer/kleur	*cijfer/kleur*	*cijfer/kleur*	*cijfer/kleur*
0 transparant	4 donkerblauw	8 rood	12 donkergroen
1 zwart	5 lichtblauw	9 lichtrood	13 magenta
2 groen	6 donkerrood	10 donkergeel	14 grijs
3 lichtgroen	7 hemelsblauw	11 lichtgeel	15 wit

Zo zal de lijn in ons voorbeeld inderdaad in de voorgrondkleur zwart (1) tegen een lichtrode achtergrond (9) met donkergroene randgebieden (12) worden afgebeeld.
Nog een aantal voorbeelden:

`COLOR 1` : maak alleen de kleur van de voorgrond zwart (1), met andere woorden teken zwarte lijnen.

`COLOR ,2` : verander de kleur van de achtergrond in groen. Merk op dat we een komma voor het getal plaatsen.

`COLOR 1,2,11` : maak de kleur van de voorgrond zwart (1), van de achtergrond groen (2) en van de randgebieden lichtgeel (11).

COLOR ,,11 : verander alleen de kleur van de randgebieden. Merk op dat voor het cijfer twee komma's zijn geplaatst.

Ten slotte merken we nog op, dat als men de achtergrondkleur wil wijzigen we na de COLOR-instructie altijd een CLS-instructie (*CL*EAR *S*C*R*EEN: wis het scherm) moeten plaatsen.

COLOR= Veel schilders maken gebruik van een palet, waarop door het mengen van kleuren een gewenste kleur kan worden verkregen. Een schilder maakt dan gebruik van een groot aantal 'dotten' verf, die hij kan mengen. Uit de theorie van de kleuren weten we echter dat we iedere kleur kunnen verkrijgen door slechts *drie* kleuren te mengen. Zo wordt bij de kleuren-TV beeldbuizen met een rode, groene en blauwe lichtstraal iedere kleur door menging verkregen. Menging wil dan zeggen dat we de intensiteit van zo'n basiskleur (rood, groen en blauw) veranderen.

Zo zijn de kleuren van de vorige tabel verkregen door de volgende mengingen uit te voeren.

Kleur Paletnr.	*Kleur*	*Intensiteit* Rood	Groen	Blauw
0	Transparant	0	0	0
1	Zwart	0	0	0
2	Groen	1	6	1
3	Lichtgroen	3	7	3
4	Donkerblauw	1	1	7
5	Lichtblauw	2	3	7
6	Donkerrood	5	1	1
7	Cyaan	2	6	7
8	Rood	7	1	1
9	Lichtrood	7	3	3
10	Donkergeel	6	6	1
11	Lichtgeel	6	6	4
12	Donkergroen	1	4	1
13	Hagenta	6	2	5
14	Grijs	5	5	5
15	Wit	7	7	7

We zien bijvoorbeeld hoe de kleur lichtrood met kleurnummer 9 is verkregen door rood, groen en blauw te mengen volgens de intensiteiten 7, 3 en 3.

Als we geen bijzondere instructies geven, zal kleurnummer 9 altijd met deze verhouding overeenkomen.

Maar het is ook mogelijk om een kleurnummer zelf te defi-

niëren en wel met de COLOR=-instructie. Deze instructie heeft de volgende vorm:

COLOR= kleurnummer, intensiteit rood, intensiteit groen, intensiteit blauw.

We voegen ter illustratie, aan het programma waarmee we de zwarte lijn tegen de lichtrode achtergrond tekenden, de volgende instructie toe:

```
15 COLOR=(9,1,2,7)
```

Zo wordt het volledige programma:

```
10 SCREEN 2
15 COLOR=(9,1,2,7)
20 COLOR 1,9,12
30 CLS
40 FOR K=1 TO 100
50 PSET (K,K)
60 NEXT K
70 GOTO 70
```

We hebben nu kleurnummer 9 zelf gedefinieerd en wel door de intensiteit-verdeling 1,2 en 7 aan te geven. Merk op dat dit werkelijk een nieuwe kleur is, een kleur die in onze standaardtabel nog niet is aangegeven!

Na het RUN-commando zien we dat onze lichtrode achtergrond nu is veranderd in een felblauwe achtergrond.

De andere kleuren d.w.z. de kleuren van de voorgrond (de zwarte lijn) en de randgebieden (groen) zijn niet veranderd. Dit klopt want de kleurnummers 1 en 12 zijn niet hergedefinieerd. Eventueel kunnen we deze nummers ook aan een zelf gekozen kleur koppelen.

Als we de oorspronkelijke kleuren d.w.z. de kleuren van onze standaardtabel weer willen zien, moeten we de instructie COLOR=NEW gebruiken.

Voeg aan het voorafgaande programma maar eens toe:

```
27 COLOR=NEW
```

We krijgen nu weer het oorspronkelijke kleurenplaatje te zien.

Een kleuraanduiding achter een grafische instructie

De PSET-instructie is een grafische instructie, omdat we er plaatjes mee kunnen opbouwen. In het volgende hoofdstuk zullen we nog de instructies LINE en CIRCLE behandelen. Ook dit zijn grafische instructies. We kunnen ook *direct* kleuraanduidingen in zo'n grafische instructie opnemen en wel:

1. door direct achter de instructie een kleurnummer te plaatsen. Hiermee wordt dan de voorgrondkleur gedefinieerd.

2. eventueel door hierna nog een zgn. logische bewerking op te geven. Zo zijn AND en OR voorbeelden van logische bewerkingen. Een volledig overzicht van alle mogelijke bewerkingen treffen we aan in het Overzicht MSX-BASIC-opdrachten.

We starten met een eerste voorbeeld:

```
10 SCREEN 5
20 COLOR 15,4,4
30 CLS
40 FOR K=1 TO 100
50 PSET (K,K),1
60 NEXT K
70 GOTO 70
```

We zien nu een zwarte lijn verschijnen, inderdaad kleurnummer 1 komt met zwart overeen.

We zien dus hoe door het plaatsen van een kleurnummer de voorgrondkleur die in de COLOR-instructie is aangegeven (voorgrond was 15) wordt overschreven.

De bewerking

Nu kunnen we nog een logische bewerking aangeven. Verander regel 50 maar eens in:

```
50 PSET (K,K),1,OR
```

We zien hoe na de kleuraanduiding nu de logische bewerking OR is aangegeven.

Om nu te bepalen welke kleur we uiteindelijk krijgen, moeten we eerst twee begrippen uitleggen, namelijk de begrippen SCREEN COLOR (SC) en DESTINATION COLOR (DS). SC is steeds de voorgrondkleur zoals door PSET bepaald, waarmee wordt gewerkt en DS de achtergrondkleur, zoals die op het scherm aanwezig is. De OR-bewerking wordt hierbij gedefinieerd als een optelling van de twee getallen.

Nu is bij dit voorbeeld SC gelijk aan 1 en DS gelijk aan 4. De term OR levert hier dus de waarde 1+4=5, met andere woorden lichtblauw.

3 GRAFISCHE VOORSTELLINGEN: LINE, DRAW, CIRCLE, PAINT EN COPY

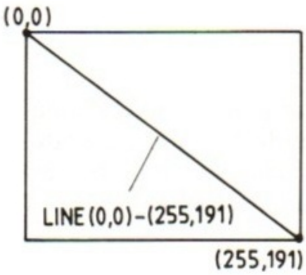
Inleiding In dit hoofdstuk behandelen we de overige mogelijkheden om lijnen op het beeldscherm weer te geven. De instructies LINE en DRAW bieden de mogelijkheid om op eenvoudige wijze rechte lijnen aan te geven, terwijl we met CIRCLE bogen (onderdelen van een cirkel of een ellips) op het scherm kunnen tekenen.

Al deze instructies gaan uit van het grafische scherm, zoals dit in het vorige hoofdstuk besproken werd.

LINE De meest eenvoudige vorm van de LINE-instructie is:

```
LINE (X1,Y1)-(X2,Y2)
```

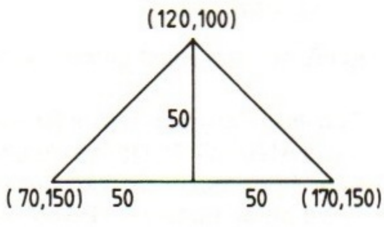
Met (X1,Y1) wordt dan het beginpunt van de lijn opgegeven en met (X2,Y2) het eindpunt.
Voorbeeld:




```
10 SCREEN 2
20 LINE(0,0)-(255,191)
30 GOTO 30
```

Als resultaat zien we nu een schuine lijn over het beeldscherm lopen.

In het nu volgende programma zullen we een driehoek tekenen en wel volgens onderstaande definitie:



Programma:

```
10 SCREEN 2
20 LINE ( 70,150)-(170,150)
30 LINE (170,150)-(120,100)
40 LINE (120,100)-( 70,150)
50 GOTO 50
```

Eventueel kunnen we de beginpositie weglaten. In dit geval wordt als beginpunt het eindpunt genomen dat aan de hand van een vorige instructie werd bereikt. Zo verkrijgen we dezelfde driehoek met het volgende programma:

```
10 SCREEN 2
20 LINE ( 70,150)-(170,150)
30 LINE -(120,100)
40 LINE -( 70,150)
50 GOTO 50
```

We kunnen bovendien een kleur aan een lijn opgeven en wel volgens dezelfde methode als bij PSET, kortom door het plaatsen van een komma en een kleurcode, dus:

```
LINE (X1,Y1)-(X2,Y2), kleurcode
```

We kunnen met de LINE-instructie eenvoudig blokken tekenen. We breiden de instructie daartoe uit met ,B dus:

```
LINE (X1,Y1)-(X2,Y2), kleur ,B
```

Bijvoorbeeld:

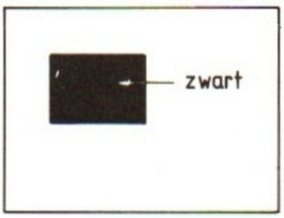
```
10 SCREEN 2
20 LINE (50,50)-(100,100),1,B
30 GOTO 30
```



geeft het hiernaast getoonde resultaat:

Merk op dat we het blok kennelijk volledig gedefinieerd hebben door twee schuin tegenover elkaar liggende hoekpunten in de LINE-instructie aan te geven. Als we in plaats van B nu BF invullen, wordt het gehele vierkant ingekleurd, bijvoorbeeld:

```
10 SCREEN 2
20 LINE (50,50)-(100,100),1,BF
30 GOTO 30
```

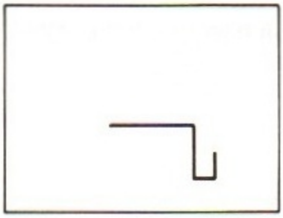


geeft het hiernaast getoonde resultaat:

Ten slotte kunnen we na BF nog een logische operatie plaatsen zoals AND of OR. Op deze wijze kan men dan de kleur beïnvloeden. Een voorbeeld van het effect van zo'n logische operatie werd bij de instructie PSET behandeld. In het Overzicht MSX-BASIC opdrachten ziet men bij de instructie PSET een volledig overzicht van de verschillende mogelijkheden die men met logische uitdrukkingen kan bereiken.

DRAW De DRAW-instructie is een bijzonder krachtige instructie om horizontale, verticale en schuine lijnen te tekenen.

We geven een eenvoudig voorbeeld:



```
10 SCREEN 2
20 PSET (127,95)
30 DRAW "R50D50R25U25"
40 GOTO 40
```

Het resultaat is hiernaast weergegeven.

Regel 20 geeft het startpunt van onze tekenoefening. De daarop volgende DRAW-instructie geeft door middel van een string aan wat getekend moet worden en wel volgens:

R50 teken een lijn door 'de pen' 50 punten naar rechts te trekken.

D50 ga nu 50 punten naar beneden (D komt van 'down' hetgeen wil zeggen 'naar beneden').

R25 ga vervolgens weer 25 punten naar rechts.

U25 en vervolgens 25 punten naar boven (U komt van 'up' hetgeen wil zeggen 'naar boven')

Een volledig overzicht van de mogelijkheden van DRAW wordt in het Overzicht MSX-BASIC-opdrachten gegeven.

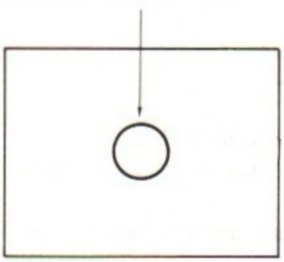
CIRCLE Dit is de laatste instructie die we bespreken om lijnen, in dit geval curven, te tekenen. De algemene vorm is:

`CIRCLE (X,Y) ,` straal, kleur, beginhoek, eindhoek, aangezichtsverhouding

met:

(X,Y)	coördinaten van middelpunt
straal	waarde voor de straal
kleur	code voor de kleur
beginhoek	hoek vanaf waar de boog getekend wordt (opgeven in radialen: $0..2\pi$)
eindhoek	hoek met betrekking tot waar de boog wordt getekend (opgeven in radialen: $0..2\pi$)
aangezichts-verhouding	getal dat opgeeft in hoeverre de cirkel afgeplat, dat wil zeggen als een ellips wordt getekend. Vult men hier de waarde 1.4 voor in, dan verkrijgt men een cirkel in SCREEN-modus 2, 3 en 4. Vult men hier de waarde 1,36 voor in, dan verkrijgt men een cirkel in SCREEN-modus 5 en 8. Gebruikt men de waarde 0.68, dan verkrijgt men een cirkel in SCREEN-modus 6 en 7.

zwarte cirkel met middelpunt (127,95) en straal R=50



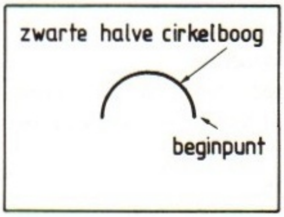
We geven enkele voorbeelden:

```
10 SCREEN 2
20 CIRCLE (127,95),50,1,,,1.4
30 GOTO 30
```

Hiernaast ziet u het resultaat.

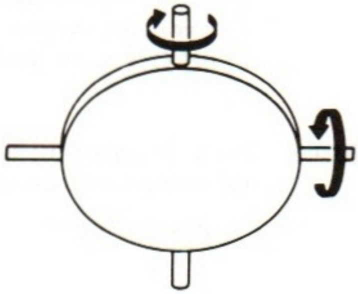
Het effect van de begin- en eindhoek kunnen we met behulp van het volgende programmaatje uitzoeken:

```
10 INPUT B
20 INPUT E
30 SCREEN 2
40 CIRCLE (127,95),50,1,B,E,1.4
50 GOTO 50
```



Als men nu voor B de waarde 0 en voor E de waarde 3.1415 invult, verkrijgt men de hiernaast afgebeelde halve cirkel:

Het laatste getal van de CIRCLE-instructie bepaalt de zgn. aangezichtsverhouding. We kunnen hierbij het beste aan het volgende beeld denken:



Als de schijf om één van de assen draait, zullen we de schijf steeds onder een andere hoek, dat wil zeggen aangezichtsverhouding zien.

We kunnen het effect het beste aan de hand van het volgende programmaatje bestuderen:

```
10 INPUT K
20 SCREEN 2
30 CIRCLE (127,95),50,1,,,K
40 GOTO 40
```

PAINT De volgende instructie die we in dit hoofdstuk bespreken, is de PAINT-instructie. Hiermee kunnen we bepaalde vlakken inkleuren en wel door een willekeurig punt in zo'n in te kleuren vlak aan te wijzen.
De algemene vorm luidt:

`PAINT (X,Y),` kleurvlak, kleurlijn

met

(X,Y)	coördinaten van het aan te wijzen punt
kleurvlak	kleur waarmee het vlak moet worden ingekleurd
kleurlijn	kleur waarmee de grens van het vlak moet worden aangegeven

Gewoonlijk laten we de laatste code weg, omdat voor het grafische scherm geldt dat de kleur van de grenslijn gelijk moet zijn aan de kleur van het vlak. Aangezien we altijd de grenzen van het vlak moeten aangeven, alvorens we kunnen inkleuren, ligt de code voor de kleur in feite ook vast.

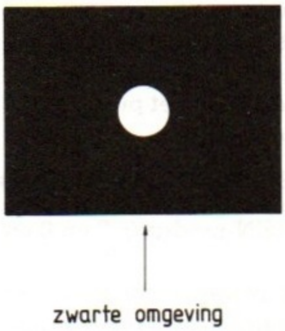
Een voorbeeld:

```
10 INPUT X,Y
20 SCREEN 2
30 CIRCLE (127,95),50,1,,,1.4
40 PAINT (X,Y),1
50 GOTO 50
```



Als we met PAINT nu een punt binnen de cirkel aanwijzen, krijgen we het hiernaast afgebeelde resultaat:

We zien hoe nu het vlak binnen de cirkel wordt ingekleurd. Als we echter een punt buiten de cirkel aanwijzen, krijgen we:



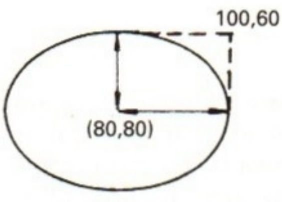
In dit geval wordt dus het andere gebied gekleurd.

We plaatsen hierbij nog de opmerking, dat in SCREEN-modus 5, 6, 7 en 8 de kleur van de kleurlijn mag verschillen van de kleur van het 'kleurvlak'.
Voorbeeld:

```
10 SCREEN 5
20 CIRCLE (127,95),50,1,,,1.36
30 PAINT (127,95),8,1
40 GOTO 40
```

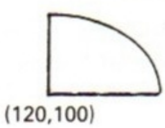
COPY

COPY wil zeggen kopiëren en dat is dan ook de functie van deze zeer krachtige instructie. Er zijn twee mogelijkheden: of we maken een kopie van een gedeelte van het scherm en beelden dat af op een ander gedeelte van het scherm, of we maken een kopie van het scherm en slaan deze in een array op. Bedenk daarbij wel dat COPY alleen gebruikt mag worden in SCREEN-modus 5 t/m 8.



We starten direct met een voorbeeld:

```
10 SCREEN 5
20 CIRCLE (80,80),20,15
30 PAINT (80,80),1,15
40 COPY (80,80)-(100,60) TO (120,100)
50 GOTO 50
```



Als we nu het scherm bekijken zien we een afgeplatte cirkel en een segment dat tevens is afgebeeld. Zie de afbeeldingen hiernaast.

Regel 10, 20 en 30 leveren geen problemen op. Regel 40 bevat de COPY-instructie. Deze heeft de volgende structuur:

```
COPY  (80,80)-(100,60) TO (120,100)
```

bepaalt te kopiëren gebied — bepaalt plaats waar afbeelding komt

We zien hoe direct na COPY het te kopiëren gebied wordt aangegeven en wel door twee punten (X1,Y1) en (X2,Y2) op te geven die zo een rechthoek bepalen. In voorgaande figuur wordt dit geïllustreerd.

We zien hoe de plaats waar het te kopiëren gedeelte moet komen te staan maar door één punt wordt aangegeven. Zoals we uit de figuur kunnen opmaken, komt dit punt dan met de linker benedenhoek van de rechthoek overeen.

In het nu volgende voorbeeld tonen we hoe een kopie in een array kan worden opgeslagen en tevens hoe we vanuit dat array weer een afbeelding op het scherm kunnen krijgen.

Het zal duidelijk zijn dat het opslaan van een deel van het beeld in een array onmiddellijk leidt tot de vraag 'hoe groot moet die array dan zijn'.

Welnu, de volgende formule, die letterlijk in het programma kan worden overgenomen, bepaalt deze grootte.

```
INT((<PIXEL>*(ABS(X2-X1)+1)*(ABS(Y2-Y1)+1)+7)/8)+4
```

Hierbij is <pixel> gelijk aan 4 in SCREEN-modus 5, 7 en 8 en gelijk aan 2 in SCREEN-modus 6.

Alhoewel de formule er ingewikkeld uitziet volstaan we met de gelukkige gedachte dat we hem slechts in het programma over moeten nemen en niet hoeven te begrijpen. Voor ons zojuist gegeven voorbeeld geldt X1=80, X2=100, Y1=80 en Y2=60. Ons programma luidt:

```
5  B=INT((4*(ABS(100-80)+1)*(ABS(60-80)+1)+7)/8)+4
6  DIM A(B)
10 SCREEN 5
20 CIRCLE (80,80),20,15
30 PAINT (80,80),1,15
40 COPY (80,80)-(100,60) TO A
50 COPY A,3 TO (120,120)
60 GOTO 60
```

Regel 5 geeft de bekende formule. In regel 6 ziet men de declaratie van de array. Regel 40 toont de betreffende COPY-instructie. Merk op dat na TO nu alleen de naam van de array wordt genoemd. Regel 50 ten slotte toont de vorm, waarbij vanuit de array de kopie naar het beeld wordt geplaatst.

Merk op dat na A de code 3 wordt aangegeven. Dit is de zgn. richtingscode, die aangeeft hoe het beeld wordt geplaatst. Voor een volledige beschrijving hiervan verwijzen we naar het Overzicht MSX-BASIC-opdrachten.

4 SPRITES

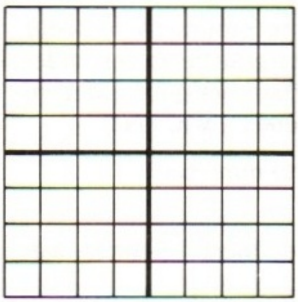
Sprite

De allereerste vraag die we ons in dit hoofdstuk stellen, is natuurlijk 'wat is precies een sprite?'. Welnu een sprite is een figuurtje dat we kunnen vastleggen met behulp van een ruitjespatroon. Een dergelijke figuur geven we vervolgens een nummer. Daarna kunnen we aan de hand van een speciale instructie allerlei manipulaties met zo'n figuurtje uitvoeren. De belangrijkste manipulatie is dat we zo'n figuurtje eenvoudig over het scherm kunnen verplaatsen.

Uiteraard kunnen we ook met PRINT- en PSET-instructies allerlei figuurtjes definiëren. Voor het verplaatsen van dergelijke met PRINT- en PSET-instructies opgebouwde figuurtjes zijn echter relatief veel instructies nodig, met als resultaat dat een snelle verplaatsing van een zo opgebouwde figuur in feite niet mogelijk is.

Het werken met sprites geeft dus wezenlijk meer mogelijkheden. Het vormt met name een krachtig gereedschap om bijvoorbeeld packman-achtige wezens over het scherm te laten dwalen.

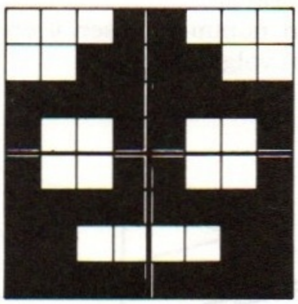
Het vastleggen van een sprite



In het nu volgende zullen we tonen hoe we zo'n figuurtje, met andere woorden zo'n sprite, definiëren. Hierbij gaan we uit van een 8×8 matrix.

Vervolgens kleuren we dan bepaalde vakjes zwart om de figuur te verkrijgen die we wensen.

Om bijvoorbeeld een spookje te verkrijgen, kunnen we vakjes als volgt inkleuren:



0001 : 1000	18
0011 : 1100	3C
1111 : 1111	FF
1001 : 1001	99
1001 : 1001	99
1111 : 1111	FF
1100 : 0011	C3
1111 : 1111	FF

Links zien we het spookje en in de linkerkolom ziet men hoe we met behulp van enen en nullen (zwart en wit) dezelfde figuur kunnen omschrijven.

De stippellijn is een 'hulplijn'; deze splitst iedere regel in twee rijtjes van vier enen en/of nullen. De bovenste regel bestaat bijvoorbeeld uit de reeksen 0001 en 1000.

In de rechterkolom ziet men een notatie met andere codes.

Deze codes kan men direct uit de volgende tabel afleiden. (N.B. het gaat hier om de zgn. hexadecimale notatie).

reeks	*code*	*reeks*	*code*	*reeks*	*code*	*reeks*	*code*
0000	0	0100	4	1000	8	1100	C
0001	1	0101	5	1001	9	1101	D
0010	2	0110	6	1010	A	1110	E
0011	3	0111	7	1011	B	1111	F

Met behulp van de zo gevonden codes definiëren we nu in een programma de sprite en wel met de volgende instructie:

SPRITE$ (nummer)= reeks CHR$-instructies

Voor ons voorbeeld:

```
SPRITE$(1)=CHR$(&H18)+CHR$(&H3C)+CHR$(&HFF)+
           CHR$(&H99)+CHR$(&H99)+CHR$(&HFF)+
           CHR$(&HC3)+CHR$(&HFF)
```

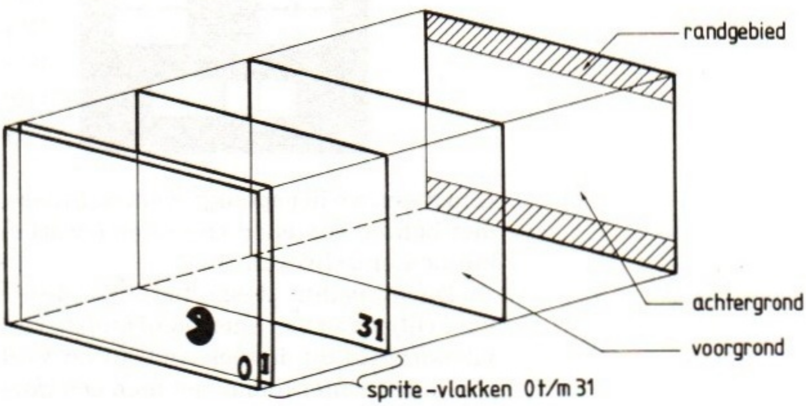
Merk op dat hier de reeks codes 18, 3C, FF, etc. in CHR$-instructies zijn opgenomen, waarbij iedere code wordt voorafgegaan door de tekens &H.

PUT SPRITE Op de zojuist besproken wijze is onze sprite (nummer 1) volledig vastgelegd en we kunnen vervolgens het figuurtje op een willekeurige plaats op het scherm plaatsen. Hiervoor gebruiken we de PUT SPRITE-instructie, die de volgende vorm heeft:

PUT SPRITE vlak, (x,y), kleur, sprite-nummer

Hierin is:

vlak een nummer tussen 0 en 31. Men moet hierbij denken aan een volgende opzet:



De sprites bewegen zich als het ware over doorzichtige glazen schermen, die op bovenstaande wijze achter elkaar zijn geplaatst. De ligging van een vlak bepaalt welke sprite wordt afgedekt als twee sprites elkaar overlappen.

(x, y) de coördinaten van de positie waar de sprite moet worden afgebeeld.

kleur het getal waarmee de kleur van de sprite wordt aangegeven.

nummer nummer dat aan de sprite gegeven is en wel aan de hand van de SPRITE$ (nummer)-instructie.

ON SPRITE GOSUB en SPRITE ON

Bij veel spelletjes is het belangrijk om te weten of op een gegeven moment twee sprites elkaar overlappen, of populair gezegd, of er een botsing optreedt.

Om dit te kunnen constateren, biedt MSX-BASIC de ON SPRITE GOSUB-instructie. Deze heeft steeds de volgende vorm:

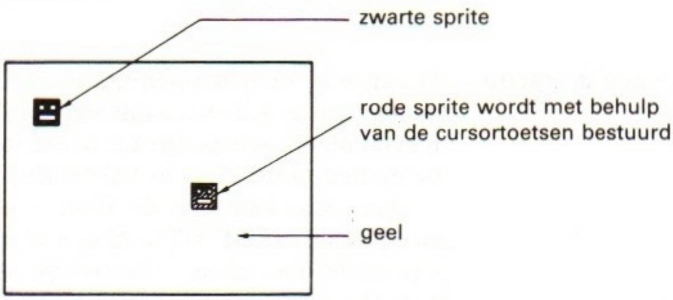
`ON SPRITE GOSUB` regelnummer

Als de computer nu constateert dat twee sprites elkaar overlappen, wordt naar een subroutine via het aangegeven regelnummer gesprongen.

Het is dan wel noodzakelijk dat we de waakzaamheid van de computer daartoe activeren. Dat gebeurt met behulp van de SPRITE ON-instructie.

Voorbeeld

Het volgende programma toont een voorbeeld. Op het scherm komen twee spookjes te staan. Het ene (zwarte) spookje staat stil, terwijl het andere (rode) spookje door middel van de cursortoetsen bestuurd wordt. De functie STICK wordt beschreven in het Overzicht MSX-BASIC-opdrachten.
In beeld:



Het volledige programma luidt:

```
10 CLS
20 COLOR,11,11
30 SCREEN 2
40 X=100:Y=100
```

```
 50 FOR K=1 TO 2
 60 SPRITE$(K)=CHR$(&H18)+CHR$(&H3C)+CHR$(&HFF)+
              CHR$(&H99)+CHR$(&H99)+CHR$(&HFF)+
              CHR$(&HC3)+CHR$(&HFF)
 70 NEXT K
 80 SPRITE ON
 90 PUT SPRITE 0,(40,40),1,1
100 D=STICK(0)
110 IF D=1 THEN Y=Y-1
120 IF D=2 THEN X=X+1:Y=Y-1
130 IF D=3 THEN X=X+1
140 IF D=4 THEN X=X+1:Y=Y+1
150 IF D=5 THEN Y=Y+1
160 IF D=6 THEN X=X-1:Y=Y+1
170 IF D=7 THEN X=X-1
180 IF D=8 THEN X=X-1:Y=Y-1
190 PUT SPRITE 1,(X,Y),6,2
200 ON SPRITE GOSUB 220
210 GOTO 100
220 PLAY "CCC"
230 RETURN
```

Regel 10 t/m 30 spreken voor zich. Regel 40 bepaalt de beginpositie van de rode sprite. Hierna worden in regel 50 t/m 70 twee sprites gedefinieerd, die overigens precies dezelfde vorm hebben (namelijk de reeds besproken vorm).

Hierna wordt de computer in de toestand 'attent op botsingen' geplaatst en wel met de SPRITE ON-instructie (regel 80). Regel 90 plaatst de zwarte sprite op het scherm en wel op positie (40,40). Vervolgens ziet men de constructie om de stand van de cursortoetsen uit te lezen. Het resultaat wordt dan meegegeven aan de PUT SPRITE-instructie van regel 190. Hiermee wordt dan de rode sprite geplaatst.

Regel 200 toont de 'botsings-proef'; als er een botsing optreedt, springt de computer naar de subroutine van regel 220.

Kleur van de sprite

We zagen bij de behandeling van PUT SPRITE hoe we de kleur van een sprite kunnen aangeven. MSX2 stelt de gebruiker ook in staat meerkleurige sprites te definiëren. Hiervoor gebruiken we de instructie COLOR SPRITE$.

Daarnaast kan men de kleur ook nog bepalen door de instructie COLOR SPRITE. Alleen de laatst toegepaste instructie bepaalt de kleur die we uiteindelijk zien. In het Overzicht MSX-BASIC-opdrachten worden COLOR SPRITE en COLOR SPRITE$ uitgebreid behandeld. Tevens worden er voorbeelden bij gegeven.

Opmerkingen We plaatsen bij het werken met sprites ten slotte nog de volgende opmerkingen:

- We kunnen de sprite een factor 2 vergroten en wel door regel 30 te vervangen door: SCREEN 2,1
- We mogen sprites ook definiëren in een matrix van 16×16. Een dergelijke matrix bestaat dus uit 4 blokken van een 8×8-matrix. Deze blokken worden als volgt genummerd

1	3
2	4

De eenvoudigste manier is dan om in vier strings dit patroon vast te leggen:

```
A$=CHR$( )+ etc.        (definitie blok 1)
B$=CHR$( )+ etc.        (definitie blok 2)
```

en evenzo C$ en D$ en vervolgens

```
SPRITE$(1)=A$+B$+C$+D$
```

In het geval dat men met 16×16-sprites werkt, gebruikt men de instructie SCREEN 2,2 of SCREEN 2,3 in het geval men een vergroting met een factor 2 wenst.

- We mogen maximaal 256 sprite-patronen met behulp van een 8×8-matrix definiëren en 64 patronen met behulp van een 16×16-matrix. Bedenk echter dat op elk sprite-vlak hoogstens één sprite mag worden weergegeven.

Ten slotte merken we nog op dat in SCREEN-modus 2 en 3 er maximaal 4 sprites naast elkaar op het scherm mogen worden geplaatst. In SCREEN-modus 4, 5, 6, 7 en 8 mogen er 8 sprites maximaal naast elkaar worden afgebeeld.

5 OVER BESTANDEN, DE MEMORY-DISK EN DE KLOKCHIP

Inleiding Een bestand (Engels: file) is een algemene benaming voor een gebundelde hoeveelheid gegevens. Zo'n gebundelde hoeveelheid gegevens kan bijvoorbeeld betrekking hebben op een hoeveelheid adressen (een adressenbestand), maar ook op een BASIC- of een machinecode-programma.

Wat de inhoud van zo'n file ook moge zijn, de file draagt in ieder geval een unieke naam, waarmee we de file dan in instructies of commando's kunnen aangeven. Vergelijk de toestand maar eens met een boekenkast. Ieder boek bevat een hoeveelheid gegevens en kan dus worden opgevat als een file. De titel van het boek komt dan overeen met de naam van de file.
Waarvoor worden files gebruikt?
Welnu, als we bijvoorbeeld over een diskdrive beschikken, zullen we onze programma's altijd als file op diskette opslaan.

Maar dat is zeker niet de enige mogelijkheid, die MSX ten aanzien van files biedt.

In het nu volgende wordt een korte opsomming gegeven van de verschillende mogelijkheden, die ons systeem biedt, te zamen met een korte aanduiding die we in instructies moeten opnemen.

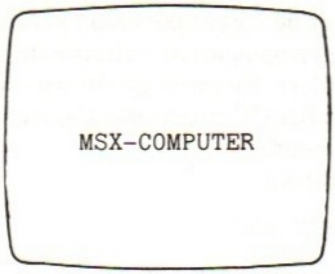
CAS De cassetterecorder.
De cassetterecorder kan worden gebruikt als goedkoop opslagmedium voor programma's en gegevens.
Een uitvoerige beschrijving wordt in Appendix A gegeven.

GRP Het grafische beeldscherm.
Het ligt niet erg voor de hand om het beeldscherm als opslagmedium voor files te beschouwen. Nu is het inderdaad zo dat we het beeldscherm niet gebruiken overeenkomstig bijvoorbeeld de cassetterecorder. Toch is het belangrijk om te weten dat we GRP in bepaalde instructies mogen gebruiken en wel omdat zo de mogelijkheid wordt geboden tekst op een grafisch scherm weer te geven.

We geven nu vast een eenvoudig voorbeeld. Later zullen we zien wat de verschillende specifieke instructies in dit programma betekenen.

```
10 SCREEN 2
20 OPEN "GRP:TEST" FOR OUTPUT AS #1
30 A$="MSX-COMPUTER"
40 PRESET (50,50)
50 PRINT #1,A$
60 GOTO 60
```

Na het RUN-commando zien we:



Kortom, we zien tekst (namelijk: *MSX-COMPUTER*) op een grafisch scherm (SCREEN 2)!

CRT Het tekst-beeldscherm.
Evenals het grafische beeldscherm kunnen we ook het 'tekstbeeldscherm' als opslagmedium voor files beschouwen.

LPT Printer.
De printer geldt met name als een randapparaat om files naar uit te voeren.
Een uitvoerige beschrijving wordt in appendix B gegeven.

COM RS232C-interface.
Met deze interface kunnen we onze computer bijvoorbeeld aan een andere computer koppelen om zo berichten over en weer te zenden. In plaats van 'berichten over en weer te zenden' spreken we ook wel van *com*municatie, vandaar de aanduiding COM.
Een uitvoerige beschrijving wordt in appendix E gegeven

A t/m F Diskdrive A t/m F.
Daar waar we in specifieke file-instructies één van de letters A t/m F opgeven, gaan we er vanuit dat de aangegeven diskdrive wordt bedoeld.
De diskdrive vormt het meest professionele opslagmedium voor files. Het is het enige medium waarbij we gegevens op een vooraf aangegeven wijze kunnen opslaan.
Een uitvoerige beschrijving wordt in Appendix D gegeven.

MEM Memory-disk.
Een gedeelte van het geheugen wordt gereserveerd als zogenaamde 'memory-disk'. Dit houdt in dat we dit gedeelte kunnen aanspreken alsof het om een diskdrive gaat.

De naam van een file De eigenlijke naam van een file mag uit maximaal 8 characters bestaan (met uitzondering van de cassetterecorder: maximaal 6!). Voorbeelden van namen zijn:

```
TEST, TEST1 en EXP1
```

Naast letters en cijfers zijn ook nog de volgende tekens toegestaan:

```
$ & # % @ ( )_ \ ^ {} ~ en !
```

Achter de eigenlijke naam plaatsen we eventueel nog een punt en een zogenaamde extensie (met uitzondering van de cassetterecorder). Hiermee geven we dan aan op wat voor *soort* gegevens (BASIC-programma's, algemene data enz.) de file betrekking heeft.
Voorbeeld:

```
TEST.BAS
```

Deze naam geeft aan dat de file TEST betrekking heeft op een BASIC-programma.

Gewoonlijk gebruiken we de volgende extensies voor filenamen op een diskette:

extensie	*betekenis.*
ASC	file, bestaande uit ASCII-codes
ASM	assembler-programma
BAK	z.g. 'back up file' (copy van een file)
BAS	BASIC-programma
COM	z.g. 'utility programma'
TEX	tekst-file
PIC	z.g. picture-file

(N.B. de volgende namen voor extensies zijn verboden: AUX, CON, LST, PRN en NUL).

Soms willen we bij bepaalde commando's niet één, maar een *verzameling* programma's aangeven. We gebruiken dan speciale tekens; de z.g. 'wild cards'.
Zo wil:

```
TEST.*
```

zeggen; 'alle programma's met de eigenlijke naam TEST en een willekeurige extensie'. Als we aangeven:

```
*.BAS
```

dan geven we daarmee aan; 'alle programma's met de extensie BAS'.
Nog een voorbeeld:

```
TEST*.BAS
```

wil zeggen; 'alle programma's met de extensie BAS, waarbij de eerste 4 characters van de eigenlijke naam bestaan uit TEST'.

Naast het teken * dat we als wild card mogen gebruiken, mogen we ook het vraagteken ? als wild card gebruiken. Zo heeft de file-aanduiding

```
TEST?
```

betrekking op alle files waarvan de eerste 4 letters overeenkomen met TEST en het vijfde character willekeurig mag zijn. Be-

schikken we bijvoorbeeld over de files TEST1, TEST2, TEST3 dan worden met TEST? al deze files aangeduid.

De OPEN-, PRINT#- en CLOSE-instructie

Als we met files willen werken, moeten we eerst een naam en een nummer aan de file geven. Dit gebeurt dan met behulp van de OPEN-instructie. Hierna kunnen we de sequentiële file vullen met behulp van PRINT#-instructies. Tenslotte dienen we als we in een programma met files werken het programma altijd af te sluiten met een CLOSE-instructie.

Tot zover een droge opsomming van de instructies, die nodig zijn om een file te benoemen en van gegevens te voorzien.

Voor een beginnende gebruiker zal dit alles echter zeker niet zo duidelijk zijn. We zullen nu aan de hand van een voorbeeld de zojuist opgenoemde instructie uitvoerig illustreren.
Voorbeeld:

```
10 REM SCHRIJVEN FILE
20 A$="AAP"
30 B$="NOOT"
40 C$="MIES"
50 OPEN "CAS:DATA" FOR OUTPUT AS#1
60 PRINT #1, A$
70 PRINT #1, B$
80 PRINT #1, C$
90 CLOSE #1
```

De regels 10 t/m 40 geven geen problemen. In regel 50 treft men de OPEN-instructie aan. Deze heeft de volgende vorm:

`OPEN`"naam opslagmedium: naam file"`FOR OUTPUT AS`#nummer van file

We zien in het voorgaande voorbeeld dat als naam van het opslagmedium de term CAS is ingevuld. Dit houdt in dat we kennelijk met de cassetterecorder willen werken. De naam van de file is kortweg DATA. Het laatste gedeelte:

```
FOR OUTPUT AS#1
```

wil zeggen: 'we zullen gegevens vanuit het geheugen van de computer naar de cassette sturen' en bovendien 'deze file geven we aan met nummer #1'.

We merken hierbij op dat alle gegevens die we zo op de cassette vastleggen gezamenlijk de file 'DATA' zullen vormen op cassette en tevens dat we deze file voortaan zullen aanduiden met zijn nummer (#1).

In plaats van de term OUTPUT hadden we ook de term INPUT mogen gebruiken. In dit geval gaat de computer er vanuit, dat er gegevens vanuit het opslagmedium zullen worden ingelezen; de omgekeerde weg dus. Straks zullen we hier een voorbeeld van tonen.

De regels 60 t/m 80 tonen hoe we de strings A$, B$ en C$ met

behulp van de PRINT#-instructie in de file opslaan. Merk op dat direct achter PRINT het nummer van de file staat.

Als alle gegevens zijn ingelezen, moeten we de CLOSE-instructie uitvoeren. Na CLOSE noteert men het nummer van de file. De reden van het opnemen van de CLOSE-instructie is als volgt. Het feitelijke transport van de gegevens van computergeheugen naar opslagmedium geschiedt via een zgn. buffergeheugen. Zonder CLOSE-instructie zou het buffergeheugen wel eens niet volledig kunnen worden uitgelezen.
Nog een voorbeeld:

```
10 REM LEZEN FILE
20 OPEN "CAS:DATA" FOR INPUT AS#1
30 INPUT #1, A$, B$, C$
40 PRINT A$, B$, C$
50 CLOSE #1
```

Resultaat

```
AAP        NOOT       MIES
```

We zien dat de opbouw van dit programma praktisch gelijk is aan het vorige. Daar waar in het vorige programma OUTPUT wordt gebruikt, zien we nu INPUT staan. Het gaat nu inderdaad om het lezen van gegevens van de cassetterecorder. In plaats van PRINT#-instructies worden nu INPUT#-instructies gebruikt; we moeten immers gegevens inlezen.

De PRINT-instructie van regel 40 is alleen maar opgenomen om ons er van te vergewissen, dat de oorspronkelijke gegevens inderdaad zijn uitgelezen.

Ten slotte zullen we nog wat voorbeelden van OPEN-instructies geven zonder daarbij een volledig programma voor te schotelen.

Voorbeeld 1

```
OPEN "A:PROG.DAT" FOR INPUT AS#3
```

Het gaat hier kennelijk om een file op diskette A waar vanuit gegevens zullen worden gelezen. Het nummer van de file is 3, de naam is PROG.DAT.

Voorbeeld 2

```
OPEN "GRP:TEST" FOR OUTPUT AS#1
```

Deze instructie werd in het programma bij de inleiding gebruikt. De informatie van filenummer 1 wordt nu op het grafische scherm geplaatst.

Een volledig overzicht van de OPEN-instructie wordt in het Overzicht MSX-BASIC-opdrachten gegeven.

De wijze waarop sequentiële files worden opgeslagen

De naam sequentieel wil zeggen dat de informatie in volgorde, d.w.z. na elkaar wordt opgeslagen. In het nu volgende gaan we in op de tekens 'CR', 'LF' en ',', die als scheidingstekens worden gebruikt. We starten de illustratie aan de hand van een uiteenzetting van de PRINT-instructie.

Over de komma (CR en LF)

Als we een PRINT-instructie hebben uitgevoerd zal de computer de cursor op het *begin* van een *nieuwe* regel hebben geplaatst. Naar het begin van een regel wordt aangegeven met de afkorting CR en LF heeft betrekking op het verplaatsen naar een nieuwe regel. CR en LF hebben ieder hun specifieke ASCII-code. Deze tekens worden tevens gebruikt als scheidingstekens als d.m.v. PRINT # gegevens op cassette worden geplaatst.
Voorbeeld 1

```
10 OPEN "CAS:ADDR" FOR OUTPUT AS#1
20 A=15:B=23.5:C=10:D=25.2:E=13
30 PRINT #1, A; B; C
40 PRINT #1, D; E
50 CLOSE #1
```

In dit geval kunnen de gegevens op cassette als volgt in beeld worden gebracht

15	,	23.5	,	10	CR	LF	25.2	,	13	CR	LF

Pas op met het plaatsen van afzonderlijke strings d.m.v. een PRINT #-instructie. Hierbij dienen we steeds een komma als scheidingsteken in te voeren.
Voorbeeld 2

```
10 OPEN "CAS:NAAM" FOR OUTPUT AS#1
20 A$="JOHN":B$="ELLY"
30 PRINT #1, A$,","; B$
40 CLOSE #1
```

In beeld:

JOHN	,	ELLY	CR	LF

Ten slotte plaatsen we nog de volgende opmerkingen:

- Het maximale aantal bestanden dat tegelijkertijd geopend mag zijn, bedraagt 1 tenzij met een MAXFILES-commando een groter aantal is opgegeven (bijv. MAXFILES=3).
  Bij de diskette kunnen maximaal zes files tegelijk zijn geopend.
- In plaats van PRINT # kan men ook de vorm PRINT USING # gebruiken.
- In plaats van INPUT # kan men ook de vorm LINE INPUT # gebruiken.

Random Access files Naast sequentiële files kent MSX Random Access files. Deze zijn alleen mogelijk bij diskettes. In appendix D treft men een uitvoerige beschrijving aan.

Memory-disk MSX-BASIC gebruikt niet het volledige geheugen voor het programma. De ontwerpers van het systeem gebruikten het resterende gedeelte om de mogelijkheid van een zgn. 'memory-disk' te creëren. In feite kunt u zo een deel van het geheugen gebruiken alsof het om een disk drive gaat. Deze mogelijkheid is zeer handig als we:

- meer programma's in het geheugen willen opslaan;
- gegevens in het geheugen willen opslaan;
- bepaalde handelingen zoals het sorteren van gegevens nu op de memory-disk in plaats van de diskette willen uitvoeren, omdat dit veel tijdwinst biedt.

De memory-disk kan alleen maar worden gebruikt voor opslag van programma's en opslag van sequentiële files. Voor alle duidelijkheid: COPY is niet toepasbaar bij de memory-disk.

Initialisatie Bij gebruik van de memory-disk moet men altijd van te voren in het programma de volgende instructie opnemen:

```
CALL MEMINI
```

Let wel: dit dient men slechts eenmaal te doen.

Voorbeeld van opslag van een programma.
Stel we hebben het volgende programma in het geheugen staan:

```
10 REM DEMO
20 END
```

en we willen dit in de memory-disk opslaan.
We voeren dit als volgt uit:

a. geef het commando CALL MEMINI
b. geef het commando SAVE "MEM:DEMO"

Met het commando CALL MFILES kunt u zien dat dit programma inderdaad is opgeslagen.

Om het programma weer te laden geven we het commando LOAD "MEM:DEMO". Let er op dat de naam gelijk is aan de naam die bij het SAVE-commando is meegegeven.

Bij het plaatsen van een volgend programma in de memory-disk, dan wel elke andere handeling die de memory-disk betreft, geven we het commando CALL MEMINI *niet meer*! Als we dit wel zouden doen, zou het geheugen immers weer worden gewist.

De volgende instructies/commando's zijn van toepassing op de memory-disk en worden uitvoerig in het Overzicht MSX-BASIC-opdrachten besproken.

CALL MEMINI	voor de initialisatie van de memory-disk.
CALL MFILES	geeft een overzicht van de files.
CALL MKILL	voor het verwijderen van files.
CALL MNAME	voor het geven van een nieuwe naam aan een file.

De klokchip De MSX-computer heeft een aparte chip die wordt aangeduid als de 'Klokchip'. Deze chip heeft de bijzondere eigenschap dat we er allerlei gegevens in kunnen opslaan (dus niet alleen de tijd) die bij uitzetten van de computer toch worden bewaard.

De volgende gegevens kunnen worden vastgelegd: (een uitgebreide beschrijving van de vermelde instructies vind u in het Overzicht MSX-BASIC-opdachten)

1. de tijd — instructies die hier van toepassing zijn, zijn:
SET TIME en GET TIME. SET om de tijd in te stellen en GET om de tijd op te vragen.

2. de datum — instructies die hier van toepassing zijn, zijn:
SET DATE en GET DATE. SET om de datum in te stellen en GET om de datum op te vragen. De datum kan de volgende format hebben:
MM/DD/YY of DD/MM/YY of YY/MM/DD
afhankelijk van de MSX-Landenversie.
Met het volgende programma kunt u bepalen welk format de datum heeft:

```
10 B=PEEK(&H2B)
20 A$="00000000"+BIN$(B)
30 C$=RIGHT$(A$,8)
40 IF LEFT$(C$,3)="000" THEN PRINT "YY/MM/DD"
50 IF LEFT$(C$,3)="001" THEN PRINT "MM/DD/YY"
60 IF LEFT$(C$,3)="010" THEN PRINT "DD/MM/YY"
```

Van de volgende drie gegevens kan er slechts één worden bewaard en wel de laatste die wordt gebruikt:

3. password: Met SET PASSWORD kunnen we een 'password' aan het systeem opgeven. Wanneer het systeem daarna wordt opgestart, zal eerst het 'password' ingetoetst moeten worden.

4. prompt De 'prompt' is het woordje Ok dat u na afloop van ieder commando op het beeldscherm ziet verschijnen. Met SET PROMPT kunt u daarvoor een andere term kiezen.

5. titel Met SET TITLE kunt u ervoor zorgen dat bij het opstarten van het systeem er een door u gekozen titel op het beeldscherm verschijnt.

Van de volgende commando's worden de opgegeven parameters eveneens opgeslagen in de 'klokchip'

6. SET ADJUST — Hiermee kunt u het beeld als het ware 'verplaatsen' op het beeldscherm bijvoorbeeld als de eerste kolom op uw beeldscherm net niet zichtbaar is.

7. SET BEEP — Hiermee kunt u het geluid wat bij de BEEP-instructie gegenereerd wordt naar uw eigen inzicht aanpassen.

8. SET SCREEN — Hiermee kunt u de verschillende mogelijkheden die de beeldopbouw bepalen vastleggen in de 'klokchip' zodat bij het opstarten van het systeem de door u gewenste schermindeling als standaard wordt aangenomen.

APPENDICES

A GEBRUIK VAN DE CASSETTERECORDER

We bespreken nu kort het gebruik van de cassetterecorder. Gebruik bij voorkeur een MSX-datarecorder. Bij andere cassetterecorders kunnen nog wel eens problemen ontstaan bij het instellen van het juiste volume en de juiste toonhoogte. Dit vindt u alleen door het uit te proberen!
We gaan uit van het volgende demonstratieprogramma:

```
10 REM DEMO CASSETTE
20 END
```

Om dit programmaatje op cassette te bewaren, voeren we het volgende uit:

a. sluit de cassetterecorder aan
b. zorg dat bovenstaand programma zich in het geheugen van de computer bevindt
c. zet de cassetterecorder in de stand 'opname' en geef hierna het commando CSAVE "DEMO"
   Na afloop ziet u op het scherm 'OK' staan.
   Controle op een correcte opname is mogelijk met het commando CLOAD? "DEMO", waarbij u het vraagteken na CLOAD niet mag vergeten. Dit commando wordt gegeven onmiddellijk nadat de cassette is teruggespoeld en in de stand PLAY is gezet. In het geheugen van de computer bevindt zich dan nog steeds het oorspronkelijke programma dat nu met de opname vergeleken wordt.

Om het programma naderhand van cassette in te lezen, gebruiken we het commando:

```
CLOAD
```

Het eerste programma dat de computer 'ziet' wordt ingelezen. U kunt ook een programmanaam opgeven: het commando wordt dan CLOAD 'programmanaam'. Voor ons voorbeeld zou dat dus worden:

```
CLOAD "DEMO"
```

We plaatsen hierbij nog de volgende opmerkingen:

- Als er iets misgaat met het lezen van een cassette, dan zijn er twee mogelijkheden:

  – de computer geeft de melding 'device I/O error'
  – er volgt geen enkele mededeling.

  Spoel in dit geval de band terug en experimenteer met de volume- en toonhoogte-instelling van uw cassetterecorder, om na te gaan bij welke stand er wel correct wordt geladen. Doorgaans kunt u de volumeknop op 80% en de toonhoogte op 'max' zetten. Gebruik trouwens alleen ferro- en geen CrO_2 bandjes.

- De cassetterecorder kan worden gebruikt voor de opslag van sequentiële files en programma's.

- Sommige voorbespeelde bandjes kunnen niet gekopieerd worden. Dit ligt niet aan uw recorder, maar aan de beveiligingen die op dergelijke bandjes aanwezig zijn!

- Wanneer uw MSX-computer is voorzien van een ingebouwde diskdrive of wanneer u een diskdrive via een interface aan uw MSX-computer gekoppeld heeft, dient u bij het inlezen van voorbespeelde bandjes erop te letten dat u uw MSX-computer aanzet terwijl u de SHIFT-toets ingedrukt houdt.

Overzicht van commando's

De volgende commando's hebben betrekking op het werken met de cassetterecorder. Een uitgebreide bespreking treft u aan in het Overzicht MSX-BASIC-opdrachten.

BLOAD	voor het laden van een machinetaalprogramma.
BSAVE	voor het opslaan van een machinetaalprogramma.
CLOAD	voor het laden van een programma
CLOSE	voor het sluiten van files.
CSAVE	voor het opslaan van een programma.
INPUT#	voor het lezen van een sequentiële file.
LINE INPUT#	voor het inlezen van een sequentiële file per regel.
LOAD	voor het laden van een programma in ASCII-vorm.
MERGE	voor het invoeren van een programma.
OPEN	specificeert hoe een file wordt gebruikt.
PRINT#	voor het plaatsen van gegevens in een sequentiële file.
PRINT#USING	voor het plaatsen van gegevens in een sequentiële file, waarbij het formaat wordt aangegeven.
SAVE	voor het opslaan van een programma in ASCII-vorm.

Overzicht van functies

Deze worden wederom uitgebreid in het Overzicht MSX–BASIC-opdrachten beschreven.

EOF	einde van de file.
INPUT$#	voor het inlezen van alfanumerieke gegevens.
LINE INPUT$#	voor het inlezen van alfanumerieke gegevens per regel.
VARPTR#	voor het vinden van het adres van het file-besturingsblok.

B GEBRUIK VAN DE PRINTER

Op de computer kunnen zowel MSX- als 'niet MSX-printers' worden aangesloten. Een MSX-printer is een printer die standaard voor MSX-systemen is ontwikkeld, zodat men op geen enkel punt (aansluitingen, stuurcodes, karakterset) problemen zal ondervinden. Bij een 'niet MSX-printer' kan men deze problemen uiteraard wel tegenkomen.

Allereerst zult u de printer willen gebruiken voor het afdrukken van programmalistings (commando LLIST). U kunt de printer ook gebruiken voor het afdrukken van gegevens (LPRINT-instructie). Tenslotte kunt u de gegevens opslaan in een sequentiële file (zie voor een voorbeeld de beschrijving van MAXFILES in het Overzicht MSX-BASIC-opdrachten).

Overzicht van commando's

De volgende commando's hebben betrekking op het werken met de printer. Een uitgebreide bespreking treft u aan in het Overzicht MSX-BASIC-opdrachten.

CLOSE	voor het sluiten van files.
LLIST	voor het afdrukken van een programma.
LFILES	voor het afdrukken van de namen van de files die op een diskette staan.
LPRINT	voor het afdrukken van gegevens.
OPEN	specificeert hoe een file wordt gebruikt (vanaf een printer kan men niet inlezen, dus alleen de vorm 'OUTPUT' kan worden gebruikt).
PRINT#	voor het printen van gegevens als een sequentiële file.
PRINT# USING	voor het printen van gegevens als een sequentiële file, waarbij men tevens het formaat bepaalt.

Overzicht van functies

Deze worden wederom uitgebreid in het Overzicht MSX-BASIC-opdrachten beschreven:

VARPTR#	voor het vinden van het adres van het bestandsbesturingsblok.

C GEBRUIK VAN DE JOYSTICK-CONNECTOREN

Deze connectoren worden op de machine aangegeven met 'hand control'. Via deze connectoren kunt u een aanraakpaneel, lichtpen, muis of 'track ball' aansluiten.

Overzicht van commando's

De volgende commando's hebben betrekking op het werken met de joystick-connectoren. Een uitgebreide bespreking treft u aan in het Overzicht MSX-BASIC-opdrachten.

ON STRIG GOSUB	controleert of de vuurknop van de joystick is ingedrukt.
STRIG/ON/OFF/STOP	aktiveert de controle op de vuurknop van de joystick.

Overzicht van functies

Deze worden eveneens uitgebreid in het Overzicht MSX-BASIC-opdrachten beschreven.

PAD	geeft de status van een aanraakpaneel, lichtpen, muis of 'track ball'.
PDL	geeft de stand van een 'paddle controller'.
STICK	geeft de stand van de joystick.
STRIG	geeft aan of de vuurknop van de joystick wordt ingedrukt.

D

GEBRUIK VAN DE DISKDRIVE

Indien men beschikt over een diskdrive kan men deze gebruiken voor de opslag van gegevens en programma's. In veel gevallen wordt software geleverd op een diskette. Plaatst men deze diskette in de diskdrive, dan wordt na het aanzetten van de computer doorgaans automatisch een programma gestart. Of dit zo is hangt af van het feit of er een file op de diskette is opgeslagen die benoemd is als AUTOEXEC.BAS-file, waarover straks meer.

Als u niet wil dat een programma automatisch wordt opgestart, dan zet u eerst computer aan en na de mededeling 'OK' plaatst u de diskette in de houder. Met het commando FILES krijgt u dan een overzicht van alle files die zich op die diskette bevinden. Ook dit punt zal straks uitgebreid behandeld worden.

In het nu volgende zullen we gedetailleerd ingaan op het gebruik van de diskette. We bespreken de commando's COPY (om een file te kopiëren), FILES (om een overzicht van de aanwezige files op een diskette te krijgen) en KILL (om files te verwijderen).

Vervolgens bespreken we FORMAT (om een nog ongebruikte diskette bedrijfsklaar te maken), SAVE (om een file naar diskette weg te schrijven) en LOAD (om een file vanaf diskette in het geheugen te laden).

Ten slotte geven we aan hoe we zogenaamde 'Random Access files' kunnen gebruiken. Deze appendix wordt afgesloten met een opsomming van specifieke commando's/functies, die betrekking hebben op het werken met diskettes.

Iets over drive-aanduidingen en de commando's COPY, FILES en KILL

De houder waarin men een diskette kan plaatsen, duiden we gewoonlijk aan met de naam 'drive'. We kunnen meer dan één drive aan onze computer koppelen en op grond van dit feit kan men zich natuurlijk afvragen 'hoe geven we nu aan om welke drive het gaat?' Het antwoord is eenvoudig: we geven de drives symbolisch met letters aan. Als we bijv. twee drives hebben, geven we deze de namen A en B. Deze letters gebruiken we dan ook om aan te geven op welke drive zich een bepaalde file bevindt.
Zo wil:

```
A:TEST.BAS
```

zeggen 'het programma TEST.BAS dat zich in de diskette van drive A bevindt'.
We zullen nu eens een voorbeeld geven van een commando, waarin zowel aanduidingen van files als drives voorkomen en wel het COPY-commando. Dit is dan het commando om kopieën van files te maken.

Het commando:

```
COPY "A:TEST.BAS" TO "B:"
```

wil zeggen; 'kopieer het programma TEST.BAS, dat zich op de diskette in drive A bevindt, naar de diskette die zich in drive B bevindt'.
Nog een voorbeeld:

```
COPY "A:TEST.BAS" TO "B:EXP1.BAS"
```

wil zeggen: 'kopieer het programma TEST.BAS en sla dit onder de naam EXP1.BAS op en wel op de diskette die zich in drive B bevindt'.

Nog een voorbeeld en wel een voorbeeld dat te maken heeft met het commando dat overzichten geeft van de files die zich op een diskette bevinden. Dit commando draagt de naam FILES.
Zo wil:

```
FILES "A:"
```

zeggen; 'geef een overzicht van alle files die zich op de diskette in drive A bevinden'.
We kunnen ook een enkele naam bij het FILES-commando geven, om zo na te gaan of deze betreffende file zich wel op de diskette bevindt, b.v.:

```
FILES "A:TEST.BAS"
```

Als deze file inderdaad aanwezig is, wordt de naam TEST.BAS nogmaals op het scherm weergegeven. Is dit niet het geval, dan meldt onze computer:

```
File not found
```

wat wil zeggen dat de betreffende file niet werd aangetroffen.

Nog een voorbeeld en wel een voorbeeld dat betrekking heeft op het wissen van files. Hiervoor gebruiken we het commando KILL.
Zo wil

```
KILL "A:TEST.BAS"
```

zeggen; 'wis het programma TEST.BAS dat zich op de diskette in drive A bevindt.
Wat zou het volgende betekenen?

```
KILL "A:*.*"
```

... juist, wis alle files die zich op de diskette van drive A bevinden.

We zullen straks zien hoe we de namen A en B gebruiken als we over slechts één drive beschikken.

AUTOEXEC.BAS Heeft een file de naam AUTOEXEC.BAS dan zal deze file automatisch worden opgestart bij het aanzetten van het systeem, mits de betreffende diskette zich in drive A bevindt.

FORMAT Nu nemen we een nog onbeschreven diskette en plaatsen die *nog niet* in de houder. Zo'n onbeschreven diskette moeten we eerst een bewerking laten ondergaan, die we formatteren noemen. Deze bewerking houdt in dat er een voorbewerking plaatsvindt, die nodig is om straks files op de diskette te kunnen plaatsen.

Toets daartoe in:

```
_FORMAT
```

dan wel

```
CALL FORMAT
```

en volg de instructies die op het beeldscherm verschijnen.
Als de diskette geformatteerd is, verschijnt op het scherm:

```
Format complete
Ok
```

Om te zien of we nu inderdaad een diskette hebben, waarop we bijv. programma's kunnen opslaan, typen we een kort demonstratieprogramma in:

```
10 PRINT "MSX-DISK BASIC"
```

Om dit programmaatje op diskette op te slaan, gebruiken we het SAVE-commando. We geven het volgende commando:

```
SAVE "A:TEST.BAS"
```

d.w.z. 'save het programma onder de naam TEST.BAS op de diskette die zich in drive A bevindt'.

Na het indrukken van de return-toets snort onze drive even... zou het programma nu inderdaad op de diskette zijn opgeslagen?

We kunnen dat testen door het bestaande programma dat zich nog in het geheugen bevindt met het NEW-commando te wissen (doe dit ook!). Nu laden we het programma van diskette m.b.v. het z.g. LOAD-commando:

```
LOAD "A:TEST.BAS"
```

Na het indrukken van de return-toets zal de drive weer snorren. Als we dan na afloop het LIST-commando geven, zal duidelijk worden, dat ons programma zich nu ook in het geheugen van de computer bevindt.

We hadden, om na te gaan of de file TEST.BAS zich inderdaad op de diskette van drive A bevond, natuurlijk ook het reeds besproken FILES-commando kunnen geven, bijv:

```
FILES A:*.*
```

(ook doen!), of nog korter:

```
FILES
```

We beschikken over slechts één drive

In de meeste gevallen zal men over slechts één drive beschikken en men kan zich dan afvragen 'hoe nu te handelen?'.

Welnu, in dit geval hebben de drive-namen A en B geen betrekking op de drives, maar op de twee diskettes. We illustreren dit met het volgende voorbeeld.

Stel dat we twee diskettes hebben (A en B dus) en dat we het programma TEST.BAS van diskette A naar B willen kopiëren. Stel tevens dat diskette A zich nog in de drive bevindt.
We geven nu het volgende commando:

```
COPY "A:TEST.BAS" TO "B:"
```

Nu zal dit programma eerst in het geheugen van de computer worden geladen en vervolgens toont de computer:

```
Insert diskette for drive B
and strike a key when ready
```

In dit geval zullen we nu diskette B in de houder moeten stoppen. Als het programma niet in één keer gekopieerd kan worden, zal het in stappen moeten. De computer geeft steeds aan of we diskette A dan wel diskette B in de drive moeten plaatsen.

Nog een praktische tip: noteer op de diskette de naam, m.a.w. A of B, heus, het voorkomt veel problemen!

Sequentiële files

MSX-Disk BASIC biedt tevens de mogelijkheid om met sequentiële files te werken.

Het principe van een sequentiële file hebben we reeds uitgelegd, zodat we er nu niet verder op in zullen gaan.

Er is echter nog één punt dat nog nadere bespreking verdient en wel de APPEND-functie. Met behulp van deze functie kunnen we een sequentiële file uitbreiden, hetgeen bij de cassette niet mogelijk is.

Als we bij een bestaande file meer gegevens willen opslaan, m.a.w. als we deze file willen uitbreiden, dan geven we de volgende OPEN-instructie:

```
OPEN "naam" FOR APPEND AS#nummer
```

Stel dat we met het volgende programmaatje een kleine file hebben gemaakt:

```
10 OPEN "A:DAT" FOR OUTPUT AS#1
20 INPUT X$
30 IF X$="END" THEN CLOSE #1:END
40 PRINT #1, X$
50 GOTO 20
```

Als we op deze wijze enkele strings in A:DAT hebben opgeslagen, zal op de diskette file A:DAT aanwezig zijn.

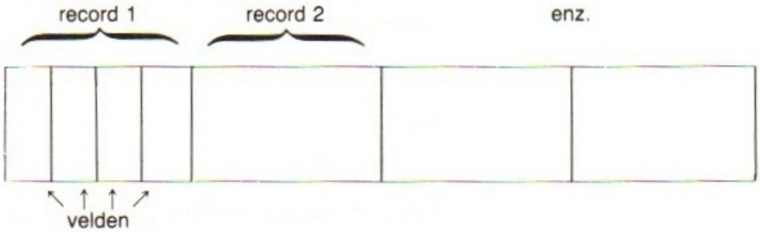
Het volgende programmaatje toont dan hoe we aan deze reeds bestaande file gegevens kunnen toevoegen:

```
10 OPEN "A:DAT" FOR APPEND AS#1
20 INPUT X$
30 IF X$="END" THEN CLOSE #1:END
40 PRINT #1, X$
50 GOTO 20
```

Het verschil met het vorige programma is in feite alleen het woord APPEND.

Random access files

Een z.g. 'random access file' is een file met een bijzondere eigenschap en wel dat we binnen bepaalde posities van de file gegevens kunnen opslaan, dan wel uitlezen. Om dit te begrijpen, kijken we naar de volgende figuur:



In deze figuur zien we een aaneenschakeling van zgn. records, terwijl ieder record op zich is onderverdeeld in zgn. velden (FIELDS).

Dit houdt in dat iedere record een vaste lengte heeft (LEN) in termen van bytes en deze lengte is opgebouwd uit velden die tevens ieder een vaste lengte hebben. Om een dergelijke random file te definiëren, moeten we eerst LEN opgeven, bijv:

```
OPEN "A:DATA" AS#1 LEN=34
```

In dit voorbeeld heeft iedere record een lengte van 34 byte. Omdat we bij deze OPEN-instructie niet de uitdrukking FOR INPUT, OUTPUT of APPEND meegeven, weet de computer automatisch dat het om een random file gaat.

Bovendien moeten we aangeven hoe ieder veld is opgebouwd en tevens met welke symbolische naam we naar zo'n veld verwijzen.
Bijvoorbeeld:

```
FIELD #1, 20 AS A$, 8 AS D$, 4 AS S$, 2 AS I$
```

We zien dat bij het voorbeeld kennelijk ieder record bestaat uit 4 velden met de namen A$, D$, S$ en I$, die dan de aangegeven hoeveelheid bytes bevatten. Het totaal aantal bytes moet uiteraard gelijk zijn aan het aantal dat we bij LEN hebben opgegeven. (controleer of dit zo is!)

Nu is als het ware het frame-work van de file vastgelegd en kunnen we kijken hoe we gegevens in zo'n file plaatsen. Hiervoor gebruiken we de instructies LSET en RSET. LSET dient steeds om strings op te slaan en RSET dient voor het opslaan van getallen.
Het volgende voorbeeld toont de verschillende mogelijkheden:

```
LSET A$ = WA$
RSET D$ = MKD$ (WD#)
RSET S$ = MKS$ (WS!)
RSET I$ = MKI$ (WI%)
```

Hierin stelt WA$ een string voor, WD# een dubbele precisie getal, WS! een enkele precisie getal en WI% een integer. Merk op dat ieder getal altijd eerst in een string omgezet moet worden met de daartoe geschikte functie.

Nu moeten we deze waarden in de file plaatsen... maar hoe? Welnu, dit hangt er vanaf waar, d.w.z. in welke record we deze gegevens willen plaatsen. Dit doen we dan aan de hand van de instructie

```
PUT #1, nummer van record
```

bijv.

```
PUT #1,3
```

Let wel, hierna zijn alle gegevens die we d.m.v. LSET en RSET aan A$, D$, S$ en I$ hebben toegekend in de file opgeslagen, zodat we op dezelfde wijze weer nieuwe gegevens aan A$, D$, S$ en I$ kunnen toekennen, die we weer in een ander record kunnen opslaan.

Ten slotte dienen we een file aan het einde van een programma altijd af te sluiten dus:

```
CLOSE #1
```

Bij het uitlezen definiëren we de random file weer op dezelfde wijze als reeds aangegeven.

Als we bij eenzelfde file in één programma, zowel schrijf- als leesacties hebben, hoeven we maar één keer de file op de bekende wijze te definiëren.
Het uitlezen gaat met de GET-instructie.
Zo zal met

```
GET #1,3
```

de inhoud van de velden van record 3 nu aan A$, D$, S$ en I$ worden toegewezen. Deze waarden kunnen we bijv. printen met;

```
PRINT A$
PRINT CVD (D$)
```

```
PRINT CVS (S$)
PRINT CVI (I$)
```

Merk de instructies CVD, CVS en CVI op om weer getallen te krijgen.

Overzicht van commando's De volgende commando's hebben betrekking op het werken met diskettes. Een uitgebreide bespreking treft u aan in het Overzicht MSX-BASIC-opdrachten.

BLOAD	voor het laden van een machinetaalprogramma of het video-geheugen.
BSAVE	voor het opslaan van een machinetaalprogramma of het video-geheugen.
CLOSE	voor het sluiten van files.
COPY	voor het kopiëren van files of gedeelten van het scherm.
DSKO$	voor het plaatsen van gegevens in een gespecificeerde sector.
FIELD	voor het verkrijgen van toegang tot de buffer die bij een random file wordt gebruikt.
FILES	geeft overzicht van de files op een diskette.
LFILES	idem, maar de files worden nu door de printer afgedrukt.
FORMAT	voor het formatteren van een diskette.
GET	voor het inlezen van een record in de buffer van een random file.
INPUT#	voor lezen van een sequentiële file.
KILL	voor het wissen van een file.
LINE INPUT#	voor het inlezen van een sequentiële file per regel.
LOAD	voor het laden van een file.
RSET, LSET	worden bij random files gebruikt.
MAXFILES	geeft maximaal aantal files aan dat geopend kan zijn (max. 6 files).
MERGE	voor het invoegen van een programma.
NAME	voor het geven van een nieuwe file-naam.
OPEN	specificeert hoe een file wordt gebruikt.
PRINT#	voor het plaatsen van gegevens in een sequentiële file.
PRINT# USING	voor het plaatsen van gegevens in een sequentiële file, waarbij het formaat wordt aangegeven.
PUT	wordt bij random files gebruikt.
SAVE	voor het opslaan van een file op diskette.
CALL SYSTEM	voor terugkeer naar MSX-DOS-systeem.

Overzicht van functies Deze worden wederom uitgebreid in het Overzicht MSX-BASIC-opdrachten beschreven.

CVI, CVS, CVD	voor het omzetten van strings naar getallen. Worden bij random files gebruikt.
DSKF	bepaalt de nog resterende ruimte op een diskette.
DSKI$	voor het laden van een sector.

EOF	einde van de file.
INPUT$#	voor het inlezen van alfanumerieke gegevens.
LINE INPUT$#	voor het inlezen van alfanumerieke gegevens per regel.
LOC	geeft de laatste locatie in de aangegeven file.
LOF	geeft de lengte van een file aan.
MKI$, MKS$, MKD$	zie CVI, CVS en CVD. Het gaat nu om het omzetten van een getal in een string.
VARPTR#	voor het vinden van het adres van het filebesturingsblok.

E GEBRUIK VAN DE RS232C-INTERFACE

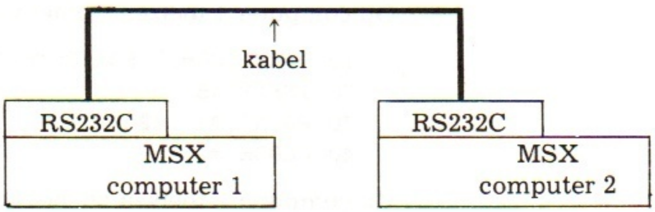
Algemeen

Communicatie is een belangrijk begrip in de computerwereld. We geven ermee aan dat we informatie van de ene naar de andere plaats overzenden. Denk maar eens aan twee computers die met elkaar in verbinding staan.

Hoe fraai is het dan niet om op eenvoudige wijze programma's dan wel gegevens over te zenden. Welnu om dit mogelijk te maken is voor de MSX-computer een zgn. RS232C-interface ontwikkeld. Dit apparaat dient er voor om over te zenden informatie in een apart daartoe geschikt formaat te brengen.

We spreken bij de RS232C van een zgn. serieel interface omdat de enen en nullen waarmee tenslotte alle gegevens zijn vastgelegd, hier na elkaar d.w.z. in serie (dus sequentieel) worden overgezonden.

Om dit proces naar behoren te laten verlopen zijn er natuurlijk wel controlesignalen nodig. In het nu volgende tonen we hoe we twee MSX-computers ieder uitgerust met een RS232C-interface met elkaar kunnen laten communiceren, dus in beeld



Voor alle duidelijkheid: in ons voorbeeld communiceert een MSX-computer met een andere MSX-computer, maar in het algemeen zou die andere computer ook een willekeurig ander apparaat kunnen zijn dat is uitgerust met een RS232C-interface (denk bijv. aan een modem om via een telefoon met een andere computer te communiceren).

De interface zelf bestaat uit een cartridge zonder kabel. Op deze cartridge bevindt zich een brede stekker (zgn. 25-pens Centronics connector).

Er zijn afzonderlijk in de handel verkrijgbare kabels die de twee cartridges van de twee computers (zie figuur) kunnen verbinden).

Het zenden van een programma

Stel we typen op computer 1 het volgende programmaatje in:

```
10 PRINT "RS232C"
20 PRINT "MSX"
```

We geven nu het volgende commando:

```
SAVE "COM:"
```

We willen dit programmaatje als het ware 'SAVEn' via de aanduiding COM waarbij COM slaat op onze RS232C-interface.
Als we nu op computer 2 intypen:

```
LOAD "COM:"
```

dan zal na het indrukken van de return-toets van beide computers het programma worden overgezonden. Let wel als we na het SAVE-commando te hebben gegeven te lang wachten met het intoetsen van het LOAD-commando krijgen we een foutmelding (Device I/O error). De reden is dat met het intoetsen van het SAVE-commando te kennen werd gegeven, dat computer 1 data wil verzenden en computer 1 kijkt dan of computer 2 'bereid is' (d.m.v. het LOAD-commando aangegeven) om die gegevens te ontvangen. Duurt dit te lang dan verschijnt de genoemde foutmelding.

In feite geldt dit voor iedere communicatie: als het ontvangende apparaat niet aangeeft 'bereid data te ontvangen' dan wordt de verbinding automatisch verbroken.

Het zenden van gegevens

Het nu volgende voorbeeld toont hoe we gegevens overzenden van computer 1 naar computer 2.
Op computer 1 toetsen we het volgende programmaatje in:

```
10 OPEN "COM:" FOR OUTPUT AS#1
20 INPUT A$
30 PRINT #1, A$
40 CLOSE #1
```

Op computer 2 toetsen we het volgende programma in:

```
10 OPEN "COM:" FOR INPUT AS#1
20 INPUT #1, A$
30 PRINT A$
40 CLOSE #1
```

Als we nu bij computer 1 na het starten van het programma de string MSX intoetsen dan verschijnt (als we het programma van computer 2 ook het commando RUN hebben gegeven) op het scherm van computer 2 tevens MSX.

Nog een voorbeeld.
Als we op beide computers het commando

```
CALL COMTERM
```

geven dan wordt alles wat op computer 1 wordt ingetypt ook zichtbaar bij computer 2 en andersom wat op 2 wordt ingetypt wordt zichtbaar op 1. In dit geval hebben we van beide computers een 'domme' terminal (typemachine om gegevens door te geven) gemaakt en kunnen we zo eenvoudig berichten uitwisselen.

Aansluiting aan een ander apparaat dan een MSX-computer (bijv. een computer van een ander merk)

Dit is in zijn algemeenheid niet te behandelen. Een en ander hangt samen met het feit dat we nu allerlei zaken moeten instellen zoals:

- de snelheid van het overzenden
- de zgn. pariteitscontrole
- de lengte van de bytes
- aantal zgn. stopbits

en vaak nog andere zaken.

Voor een beginnende gebruiker geven we dan ook een waarschuwing; het communiceren via de RS232C zal niet eenvoudig verlopen als u niet over de nodige technische kennis beschikt. Alleen als u over een duidelijke beschrijving beschikt hoe u de communicatie moet instellen, zal dit voor een eerste gebruiker uitvoerbaar zijn!

Overzicht van commando's

De volgende commando's hebben betrekking op het werken met een RS232C-interface. Een uitgebreide bespreking treft u aan in het Overzicht MSX-BASIC-opdrachten.

CALL COMINI — voor het instellen van de RS232C-interface (baudrate, lengte van 'porties', die worden verstuurd enz.).

CALL COMON
CALL COMOFF
CALL COMSTOP
CALL COM — instructies die worden gebruikt om automatisch een BASIC-subroutine uit te voeren als via de RS232C-interface een teken wordt ontvangen.

CALL COMTERM — start zogenaamde 'terminal emulator mode'.

CALL COMBREAK — voor het oversturen van zogenaamde 'break characters'.

CALL COMDTR — voor het tijdelijk uitschakelen van de RS232C-interface.

CALL COMSTAT — om bij het optreden van een fout na te gaan wat er is gebeurd.

CLOSE — voor het sluiten van de RS232C-verbinding.

INPUT# — voor het ontvangen van gegevens via de RS232C -interface.

LOAD — voor het ontvangen van een programma via de RS232C-interface.

LINE INPUT# — voor het ontvangen van gegevens via de RS232C-interface per regel.

MERGE — voor het invoegen van een programma via de RS232C-interface.

OPEN — voor het openen van de RS232C-verbinding.

PRINT# — voor het zenden van gegevens via de RS232C-interface.

PRINT# USING — voor het zenden van gegevens via de RS232C-interface, waarbij tevens het formaat wordt bepaald.

SAVE — voor het zenden van een programma via de RS232C-interface.

Overzicht van functies

Deze worden wederom uitgebreid in het Overzicht MSX-BASIC-opdrachten beschreven.

EOF — einde van de via de RS232C verzonden file.

INPUT$# — voor het ontvangen van alfanumerieke gegevens via de RS232C-interface.

LINE INPUT$# — voor het ontvangen van alfanumerieke gegevens via RS232C-interface per regel.

VARPTR# — voor het vinden van het adres van het filebesturingsblok.

F SPECIALE TEKENS EN UITDRUKKINGEN

Bewerkingstekens

Onderstaande tabel geeft een overzicht van de tekens die voor wiskundige bewerkingen gebruikt mogen worden. Het laatste cijfer geeft de prioriteitswaarde aan: uitdrukkingen worden volgens deze prioriteitswaarden uitgewerkt (van het laagste tot het hoogste cijfer). Bij gelijke prioriteitswaarde wordt de uitdrukking van links naar rechts uitgewerkt

teken	*betekenis*	*voorbeeld*	*prioriteit*
+	optellen	3+5	6
−	aftrekken	5−3	6
*	vermenigvuldigen	5×3	3
/	delen	5/3	3
∧	machtsverheffen	2^3	1
–	verandert teken	–3	4
MOD	rest bij integerdeling	12 MOD 3	5

Tekens in logische uitdrukkingen

De volgende symbolen mogen in uitdrukkingen voorkomen waarbij wordt nagegaan of ze al of niet waar zijn (bijv. na de term IF)

teken	*betekenis*	*voorbeeld*
=	is gelijk aan	IF A=B THEN...
<	kleiner dan	IF A<B THEN...
>	groter dan	IF A>B THEN...
<> of ><	ongelijk aan	IF A<>B THEN...
<= of =<	kleiner dan of gelijk aan	IF A<=B THEN...
>= of =>	groter dan of gelijk aan	IF A>=B THEN...

Termen om logische uitdrukkingen te combineren

Uitdrukkingen zoals A=B en C<>D kan men combineren met specifieke termen.

De volgende tabel geeft de verschillende termen aan met de zgn. waarheidstabellen. Hierin geeft 1 aan dat de uitspraak 'waar' is (klopt) en 0 dat de uitspraak 'onwaar' is (klopt niet, fout). Zo ziet men bijvoorbeeld bij de term AND

U1 **AND** U2

is 1 als U1 en U2 gelijk aan 1 zijn. Met U1 en U2 worden dan uit-

spraken als A=B en C<>D bedoeld. De tabel geeft in dit geval aan dat de volledige uitspraak waar is (1) als U1 en U2 ook waar zijn (1).

term	*waarheidstabel*		
NOT	U1		NOT U1
(ontkenning)	1		0
	0		1
AND	U1	U2	U1 AND U2
(logisch produkt)	1	1	1
	1	0	0
	0	1	0
	0	0	0
OR	U1	U2	U1 OR U2
(logische som)	1	1	1
	1	0	1
	0	1	1
	0	0	0
XOR	U1	U2	U1 XOR U2
(exclusieve OR)	1	1	0
	1	0	1
	0	1	1
	0	0	0
EQV	U1	U2	U1 EQV U2
	1	1	1
	1	0	0
	0	1	0
	0	0	1
IMP	U1	U2	U1 IMP U2
(logische implicatie)	1	1	1
	1	0	0
	0	1	1
	0	0	1

Bewerkingstekens bij strings

Bij strings mag alleen het teken + gebruikt worden en wel om twee strings aan elkaar te koppelen. Voorbeeld:

"AB"+"CD" wordt "ABCD"

G

FOUTMELDINGEN

Bad allocation table (60)
De diskette is niet geïnitialiseerd (niet of onjuist geformatteerd).

Bad drive name (62)
Er wordt een diskette-drive aangewezen die niet aanwezig is.

Bad file mode (61)
Men gebruikt PUT, GET of EOF met een sequentiële file, dan wel men tracht met LOAD een random file te laden. Ten slotte kan deze foutmelding optreden als de OPEN-instructie wordt gebruikt bij een file die niet correct is gedefinieerd

Bad file name (56)
De naam van het bestand (file) is onjuist. De naam voor het 'device' in een OPEN-, SAVE- of LOAD-commando is onjuist.

Bad file number (52)
Nummer van een file is groter dan toegestaan dan wel men geeft een file aan die nog niet 'geopend' is.

Can't CONTINUE (17)
Men probeert CONT-commando te geven terwijl programma niet door een fout of een STOP-instructie is onderbroken. Komt ook voor als we het programma wijzigen en onmiddellijk daarna CONT proberen.

Device I/O error (19)
Het laden van een programma of bestand gaat fout. Is de cassetterecorder wel goed ingesteld? Is de juiste I/O-poort aangegeven?

Direct statement in file (57)
In een programma dat geladen wordt komt een instructie zonder regelnummer voor, dan wel; het bestand dat geladen wordt, is geen programma.

Disk full (66)
Alle capaciteit van de diskette is benut

Disk I/O error (69)
Er wordt een fout geconstateerd bij een invoer- dan wel uitvoeractie. Dit is een zgn. 'fatal error' (fatale fout) d.w.z. dat het MSX Disk BASIC systeem door de gebruiker weer opgestart moet worden.

Disk offline (70)
Er is geen diskette in de aangegeven drive

Disk write protected (68)
Er wordt getracht iets op een 'write protected' diskette (d.w.z. een voor schrijven beveiligde diskette) weg te schrijven.

Division by zero (11)
Delen door 0 is niet geoorloofd. Deze fout kan ook optreden als men deelt door een variabele waaraan van te voren nog geen waarde is toegekend.

Field Overflow (50)
Het aantal toegekende bytes in de FIELD-instructie is groter dan 256.

File already exists (65)
De nieuwe naam van de file (d.m.v. NAME commando) is reeds een bestaande naam van een file die zich op de diskette bevindt.

File already open (54)
Er wordt getracht een file te openen die reeds geopend is.

File not found (53)
Een LOAD, KILL-commando, dan wel een OPEN-instructie verwijst naar een file die zich niet op de diskette bevindt.

File not OPEN (59)
Er wordt naar een bestand verwezen dat nog niet door een OPEN-instructie is geopend.

File still open (64)
De file is niet gesloten.

File already open (54)
Men kan geen bestand openen dat al geopend is.

Illegal direct (12)
Er wordt getracht een instructie als commando uit te voeren terwijl dat bij deze instructie niet is toegestaan.

Illegal function call (5)
Er wordt een BASIC-functie aangeroepen met een verkeerd argument.
Deze melding treedt op bij:

1 een negatieve array-index of indices die buiten het array-bereik vallen;
2 een negatief argument bij LOG of SQR;
3 een negatieve mantisse met een real exponent;
4 het gebruik van de USR-functie zonder dat een startadres is gespecificeerd;
5 een verkeerd argument bij MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR, ON...GOTO;
6 een grafisch commando wordt uitgevoerd terwijl het scherm zich niet in de grafische modus bevindt.

Input past end (55)
Er wordt getracht een gegeven in te lezen terwijl alle gegevens reeds zijn ingelezen. Kan ook optreden als het bestand leeg is, dat wil zeggen in het geheel geen gegevens bevat.

Internal error (51)
De BASIC-interpreter zelf vertoont een fout, bijvoorbeeld door beschadiging van het ROM-geheugen.

Line buffer overflow (25)
De buffer voor de invoer van één regel is vol.

Missing operand (24)
Er is iets mis met één van de gegevens die in een uitdrukking moeten worden gebruikt.

NEXT without FOR (1)
Er is een NEXT-instructie gevonden zonder bijbehorende FOR-instructie.

NO RESUME (21)
De gebruikte fout-herstel-routine bevat geen RESUME-instructie.

OUT of DATA (4)
Geprobeerd wordt met een READ-instructie gegevens te lezen die niet in een DATA-lijst zijn vastgelegd.

Out of memory (7)
Het programma is te groot voor de beschikbare geheugenruimte of het programma bevat te veel variabelen, GOSUB's of 'geneste' FOR-loops.

Out of string space (14)
De beschikbare geheugenruimte voor strings is vol. Met de CLEAR-instructie kan men deze geheugenruimte uitbreiden.

Overflow (6)
Een berekening levert een te groot getal op (zgn. overflow). Het kan ook zijn dat een opgegeven waarde buiten het toegestane bereik valt.

Redimensioned array (10)
Men probeert met DIM tweemaal dezelfde variabele te specificeren. Kan ook voorkomen als een DIM-instructie volgt op een instructie waarin reeds een array-variabele voorkomt.

Rename across disk (71)
Met het NAME-commando wordt een nieuwe file-naam toegekend waarbij de drive-aanduiding anders is dan de drive waarin zich de diskette met de betreffende file bevindt.

RETURN without GOSUB (3)
Er is een RETURN-instructie aangetroffen zonder dat via GOSUB naar een subroutine gesprongen was.

RESUME without error (22)
Er is een RESUME-instructie aangetroffen zonder bijbehorende fout-afhandelingsroutine.

Sequential I/O only (58)
Er wordt getracht de sequentiële file te behandelen als een random file.

String formula too complex (16)
De string-uitdrukking is te ingewikkeld en moet dus in kleinere stukken worden verdeeld.

String too long (15)
De string bestaat uit meer dan 255 tekens.

Subscript out of range (9)
Men gebruikt array-indexnummers die buiten het gespecificeerde bereik vallen.

Syntax error (2)
Algemene foutaanduiding: de uitdrukking voldoet niet aan de regels van BASIC.

Too many files (67)
Er wordt getracht een nieuwe file te creëren terwijl er reeds 255 files aanwezig zijn.

Type mismatch (13)
Men tracht aan een string-variabele een numerieke waarde toe te kennen of omgekeerd.

Undefined line number (8)
Er vindt verwijzing naar een niet bestaand regelnummer plaats.

Undefined user function (18)
Een FN-functie wordt aangeroepen die niet met behulp van een DEF FN-instructie van te voren is gedefinieerd.

Unprintable error (23,26-49,60-255)
Voor de aangegeven code bestaat geen standaard-foutmelding.

Verify error (20)
Er wordt een verschil geconstateerd tussen het opgenomen programma en het in het geheugen aanwezige programma.

H

ESCAPE SEQUENCES

Onder een 'Escape sequence' verstaan we een speciale code om bepaalde acties op het scherm aan te geven.

Om bijvoorbeeld de als escape sequence aangegeven code <ESC>B te geven, toetst u in:

```
PRINT CHR$(27)+"B"
```

De volgende tabel toont de verschillende mogelijkheden.

<ESC>	A	cursor één regel naar boven
<ESC>	B	cursor één regel naar beneden
<ESC>	C	cursor één positie rechts
<ESC>	D	cursor één positie links
<ESC>	H	cursor naar linker bovenhoek
<ESC>	Y	<rijnummer+32> <kolomnummer+32> cursor op aangegeven positie plaatsen
<ESC>	j	CLS
<ESC>	E	CLS
<ESC>	K	schoonmaken tot einde regel
<ESC>	J	schoonmaken tot einde scherm
<ESC>	I	schoonmaken van gehele regel
<ESC>	L	regel tussenvoegen
<ESC>	M	regel verwijderen
<ESC>	x4	verander vorm van de cursor in een vierkant
<ESC>	x5	schakel cursor uit
<ESC>	y4	verander vorm van de cursor in een streepje
<ESC>	y5	zet cursor weer aan

I DE CONTROL-CODES

Een aantal toetsen hebben in combinatie met de CTRL-toets een speciale functie. Zo zal als men de CTRL-toets indrukt en men drukt vervolgens met ingedrukte CTRL-toets op de L het scherm worden gewist. Deze actie noteren we dan met CTRL/L. Zo wil CTRL/R zeggen: op R drukken terwijl we CTRL ingedrukt houden. Het volgende overzicht toont alle mogelijkheden.

CTRL/A	de nu volgende toets roept een symbool uit de alternatieve reeks symbolen op.
CTRL/B	verplaats cursor naar vorige woord.
CTRL/C	stop AUTO-commando.
CTRL/E	alle symbolen vanaf de cursor worden gewist.
CTRL/F	verplaats cursor naar volgende woord.
CTRL/G	BEEP.
CTRL/H	komt met BS-toets overeen.
CTRL/I	komt met TAB overeen: naar volgende stop.
CTRL/J	verplaats cursor naar begin volgende regel.
CTRL/K	verplaats cursor naar linker bovenhoek.
CTRL/L	komt met CLS overeen.
CTRL/M	komt met RETURN-toets overeen.
CTRL/N	verplaats cursor naar einde van de regel (d.w.z. op de eerste plaats na het laatste symbool op die regel).
CTRL/R	komt met INS-toets overeen.
CTRL/U	wis de regel.
CTRL/\	komt met cursortoets → overeen.
CTRL/]	komt met cursortoets ← overeen.
CTRL/∧	komt met cursortoets ↑ overeen.
CTRL/–	komt met cursortoets ↓ overeen.

J OPBOUW VAN SYMBOLEN

In deze appendix wordt een overzicht gegeven van alle symbolen (characters) met hun bijbehorende codes.

Deze kunnen bijvoorbeeld in de CHR$-functie worden gebruikt. Zo kan men het teken A krijgen door PRINT "A" maar ook d.m.v. PRINT CHR$(65)

In SCREEN-modus 0 worden de twee rechter kolommen van de matrix waarmee ieder symbool wordt gedefinieerd niet afgebeeld.
Het verschil kan men zelf nagaan door te vergelijken.

```
10 SCREEN 0
20 PRINT CHR$(210)
```

en

```
10 SCREEN 1
20 PRINT CHR$(210)
```

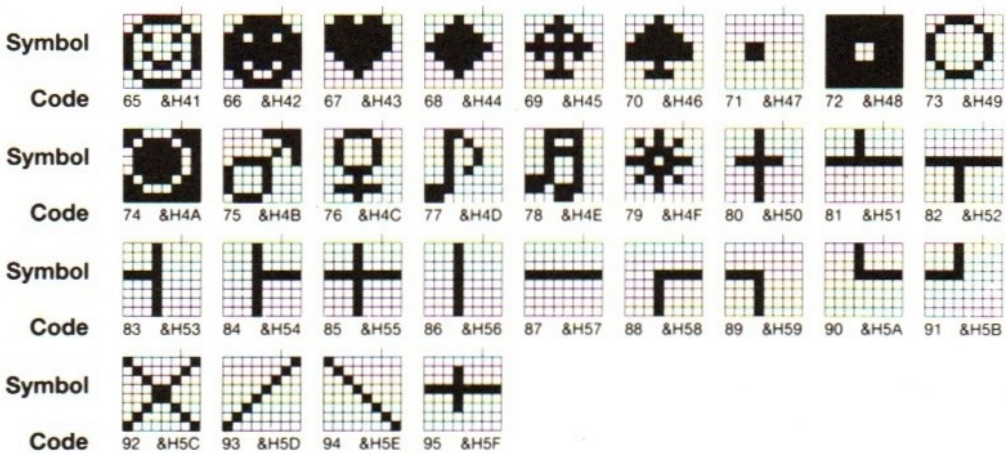
De standaardsymbolen zijn afgebeeld op de volgende pagina's.

Alternatieve symbolen

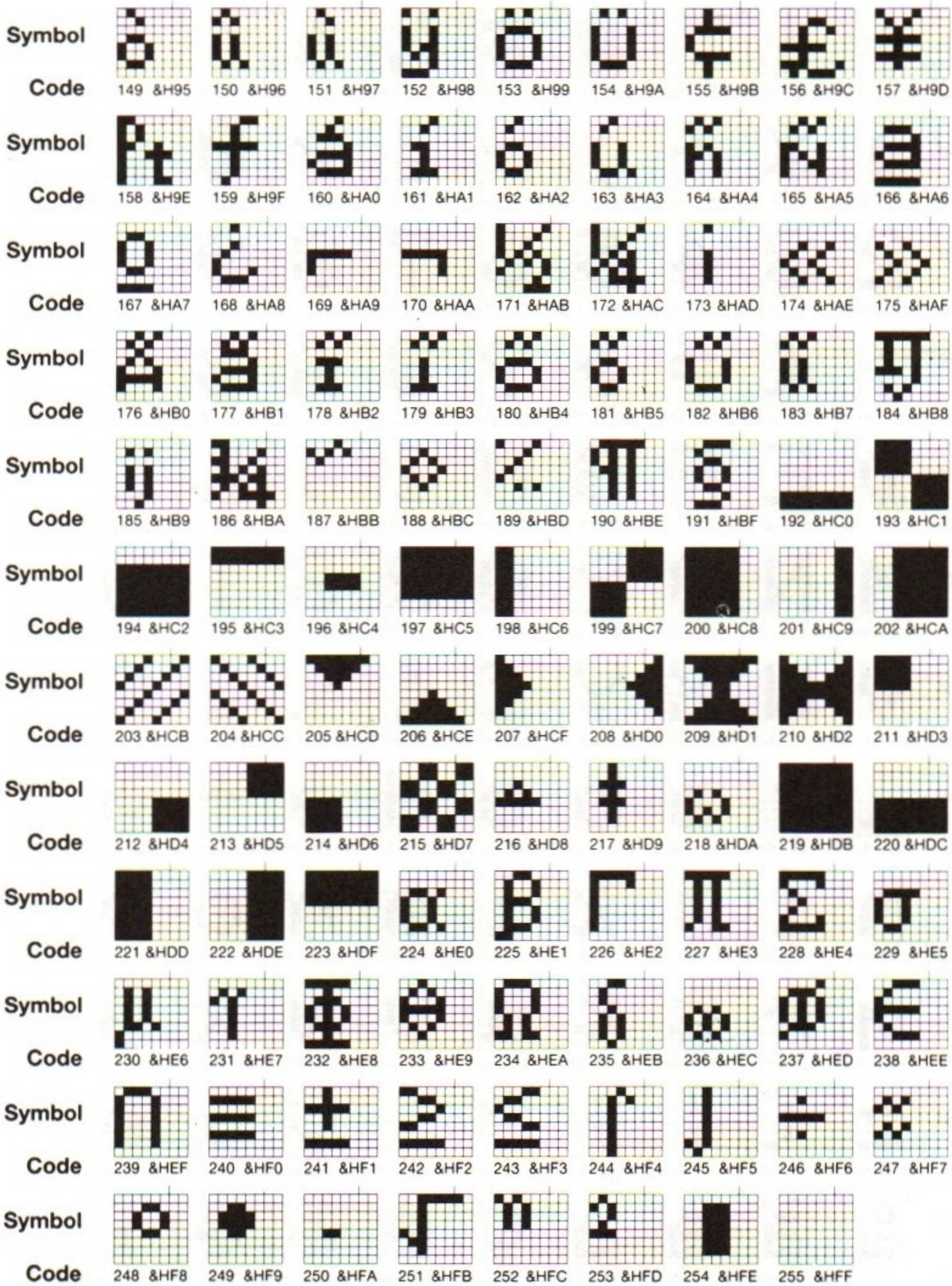
Dit zijn symbolen die men krijgt naast de reeds besproken symbolen en wel door CHR$(1) in de PRINT-instructie op te nemen. Bijvoorbeeld:

```
10 SCREEN 1
20 PRINT CHR$(65)
30 PRINT CHR$(1) + CHR$(65)
```

Regel 20 genereert een A en regel 30 heeft als resultaat dat een gezicht wordt getoond.



Symbol		!	"	#	$	%	&	'	(
Code	32 &H20	33 &H21	34 &H22	35 &H23	36 &H24	37 &H25	38 &H26	39 &H27	40 &H28
Symbol	)	*	+	,	-	.	/	0	1
Code	41 &H29	42 &H2A	43 &H2B	44 &H2C	45 &H2D	46 &H2E	47 &H2F	48 &H30	49 &H31
Symbol	2	3	4	5	6	7	8	9	:
Code	50 &H32	51 &H33	52 &H34	53 &H35	54 &H36	55 &H37	56 &H38	57 &H39	58 &H3A
Symbol	;	<	=	>	?	@	A	B	C
Code	59 &H3B	60 &H3C	61 &H3D	62 &H3E	63 &H3F	64 &H40	65 &H41	66 &H42	67 &H43
Symbol	D	E	F	G	H	I	J	K	L
Code	68 &H44	69 &H45	70 &H46	71 &H47	72 &H48	73 &H49	74 &H4A	75 &H4B	76 &H4C
Symbol	M	N	O	P	Q	R	S	T	U
Code	77 &H4D	78 &H4E	79 &H4F	80 &H50	81 &H51	82 &H52	83 &H53	84 &H54	85 &H55
Symbol	V	W	X	Y	Z	[	\	]	^
Code	86 &H56	87 &H57	88 &H58	89 &H59	90 &H5A	91 &H5B	92 &H5C	93 &H5D	94 &H5E
Symbol	_	`	a	b	c	d	e	f	g
Code	95 &H5F	96 &H60	97 &H61	98 &H62	99 &H63	100 &H64	101 &H65	102 &H66	103 &H67
Symbol	h	i	j	k	l	m	n	o	p
Code	104 &H68	105 &H69	106 &H6A	107 &H6B	108 &H6C	109 &H6D	110 &H6E	111 &H6F	112 &H70
Symbol	q	r	s	t	u	v	w	x	y
Code	113 &H71	114 &H72	115 &H73	116 &H74	117 &H75	118 &H76	119 &H77	120 &H78	121 &H79
Symbol	z	{	|	}	~		Ç	ü	é
Code	122 &H7A	123 &H7B	124 &H7C	125 &H7D	126 &H7E	127 &H7F	128 &H80	129 &H81	130 &H82
Symbol	â	ä	à	å	ç	ê	ë	è	ï
Code	131 &H83	132 &H84	133 &H85	134 &H86	135 &H87	136 &H88	137 &H89	138 &H8A	139 &H8B
Symbol	î	ì	Ä	Å	É	æ	Æ	ô	ö
Code	140 &H8C	141 &H8D	142 &H8E	143 &H8F	144 &H90	145 &H91	146 &H92	147 &H93	148 &H94



Symbol
Code
149 &H95
150 &H96
151 &H97
152 &H98
153 &H99
154 &H9A
155 &H9B
156 &H9C
157 &H9D
158 &H9E
159 &H9F
160 &HA0
161 &HA1
162 &HA2
163 &HA3
164 &HA4
165 &HA5
166 &HA6
167 &HA7
168 &HA8
169 &HA9
170 &HAA
171 &HAB
172 &HAC
173 &HAD
174 &HAE
175 &HAF
176 &HB0
177 &HB1
178 &HB2
179 &HB3
180 &HB4
181 &HB5
182 &HB6
183 &HB7
184 &HB8
185 &HB9
186 &HBA
187 &HBB
188 &HBC
189 &HBD
190 &HBE
191 &HBF
192 &HC0
193 &HC1
194 &HC2
195 &HC3
196 &HC4
197 &HC5
198 &HC6
199 &HC7
200 &HC8
201 &HC9
202 &HCA
203 &HCB
204 &HCC
205 &HCD
206 &HCE
207 &HCF
208 &HD0
209 &HD1
210 &HD2
211 &HD3
212 &HD4
213 &HD5
214 &HD6
215 &HD7
216 &HD8
217 &HD9
218 &HDA
219 &HDB
220 &HDC
221 &HDD
222 &HDE
223 &HDF
224 &HE0
225 &HE1
226 &HE2
227 &HE3
228 &HE4
229 &HE5
230 &HE6
231 &HE7
232 &HE8
233 &HE9
234 &HEA
235 &HEB
236 &HEC
237 &HED
238 &HEE
239 &HEF
240 &HF0
241 &HF1
242 &HF2
243 &HF3
244 &HF4
245 &HF5
246 &HF6
247 &HF7
248 &HF8
249 &HF9
250 &HFA
251 &HFB
252 &HFC
253 &HFD
254 &HFE
255 &HFF

K GERESERVEERDE NAMEN

De volgende woorden hebben een gereserveerde betekenis. De met een sterretje aangegeven woorden geven toekomstige uitbreidingen aan.

ABS	DEFSTR	KEY	PAINT	STRING$
AND	DELETE	KILL	PDL	SWAP
ASC	DIM	LEFT$	PEEK	TAB
*ATTR$	DRAW	LEN	PLAY	TAN
ATN	DSKF	LET	POINT	THEN
AUTO	DSKI$	LFILES	POKE	TIME
BASE	DSKO$	LINE	POS	TO
BEEP	ELSE	LIST	PRESET	TROFF
BIN$	END	LLIST	PRINT	TRON
BLOAD	EOF	LOAD	PSET	USING
BSAVE	EQV	LOC	PUT	USR
CALL	ERASE	LOCATE	READ	VAL
CDBL	ERL	LOF	REM	VARPTR
CHR$	ERR	LOG	RENUM	VDP
CINT	ERROR	LPOS	RESTORE	VPEEK
CIRCLE	EXP	LPRINT	RESUME	VPOKE
CLEAR	FIELD	LSET	RETURN	WAIT
CLOAD	FILES	*MAX	RIGHT$	WIDTH
CLOSE	FIX	MERGE	RND	XOR
CLS	FN	MID$	RSET	
*CMD	FOR	MKD$	RUN	
COLOR	*FPOS	MKI$	SAVE	
CONT	FRE	MKS$	SCREEN	
COPY	GET	MOD	SET	
COS	GOSUB	MOTOR	SGN	
CSAVE	GO TO	NAME	SIN	
CSNG	GOTO	NEW	SOUND	
CSRLIN	HEX$	NEXT	SPACE$	
CVD	IF	NOT	SPC	
CVI	IMP	OCT$	SPRITE	
CVS	INKEY$	OFF	SQR	
DATA	INP	ON	STEP	
DEF	INPUT	OPEN	STICK	
DEFDBL	INSTR	OR	STOP	
DEFINT	INT	OUT	STR$	
DEFSNG	*IPL	PAD	STRIG	

L OVERZICHT MSX BIOS-ROUTINES

Deze appendix geeft een volledig overzicht van alle BIOS-routines, zowel MSX1 als MSX2.

Het BASIC-ROM van MSX2 bestaat uit twee ROM's, namelijk een MAIN-ROM dat vergelijkbaar is met het BASIC-ROM van MSX1 en een SUB-ROM of EXTENDED ROM dat alle uitbreidingen van MSX2 ten opzichte van MSX1 bevat.

Het SUB-ROM bevindt zich in een ander slot. Om een routine in het SUB-ROM aan te roepen, moet daarom gebruik worden gemaakt van een van de speciale routines CALSLT, SUBROM of EXTROM.

In het overzicht worden eerst de routines uit het MAIN-ROM behandeld en vervolgens de routines uit het SUB-ROM. Van iedere routine wordt kort de functie omschreven, de invoer- en uitvoerparameters worden gegeven, voorzover van toepassing, en er wordt vermeld welke registers door de routine worden veranderd. Alle adressen worden gegeven in hexadecimale notatie.

Alvorens met de BIOS-routines te beginnen, eerst een overzicht van alle belangrijke RAM-adressen waarnaar in de omschrijvingen wordt verwezen:

Label	*Adres*	*Omschrijving*
	0006H	I/O adres van VDP leespoort (Video Display Processor)
	0007H	I/O adres van VDP schrijfpoort
	002DH	MSX versienummer
	409BH	warme start-adres van BASIC interpreter
	F323	beginadres van disk fout-afhandelingsroutine
TXTNAM	F3B3–F3B4	beginadres naamtabel in schermmodus 0
TXTCOL	F3B5–F3B6	beginadres kleurtabel in schermmodus 0
TXTCGP	F3B7–F3B8	beginadres patroontabel in schermmodus 0
TXTATR	F3B9–F3BA	beginadres sprite-attribuuttabel in schermmodus 0
TXTPAT	F3BB–F3BC	beginadres sprite-patroontabel in schermmodus 0
T32NAM	F3BD–F3BE	beginadres naamtabel in schermmodus 1
T32COL	F3BF–F3C0	beginadres kleurtabel in schermmodus 1
T32CGP	F3C1–F3C2	beginadres patroontabel in schermmodus 1
T32ATR	F3C3–F3C4	beginadres sprite-attribuuttabel in schermmodus 1
T32PAT	F3C5–F3C6	beginadres sprite-patroontabel in schermmodus 1
GRPNAM	F3C7-F3C8	beginadres naamtabel in schermmodus 2
GRPCOL	F3C9–F3CA	beginadres kleurtabel in schermmodus 2
GRPCGP	F3CB–F3CC	beginadres patroontabel in schermmodus 2
GRPATR	F3CD–F3CE	beginadres sprite-attribuuttabel in schermmodus 2
GRPPAT	F3CF–F3D0	beginadres sprite-patroontabel in schermmodus 2
MLTNAM	F3D1-F3D2	beginadres naamtabel in schermmodus 3
MLTCOL	F3D3–F3D4	beginadres kleurtabel in schermmodus 3
MLTCGP	F3D5–F3D6	beginadres patroontabel in schermmodus 3
MLTATR	F3D7–F3D8	beginadres sprite-attribuuttabel in schermmodus 3

Label	*Adres*	*Omschrijving*
MLTPAT	F3D9–F3DA	beginadres sprite-patroontabel in schermmodus 3
CNSDFG	F3DE	functietoets display aan/uit
FORCLR	F3E9	voorgrondkleur
BAKCLR	F3EA	achtergrondkleur
BDRCLR	F3EB	schermrandkleur
ATRBYT	F3F2	code voor de werkkleur
PRTFLG	F416	vlag: uitvoer naar printer of beeldscherm
VALTYP	F663	type van de laatst geëvalueerde expressie
AUTFLG	F6AA	vlag voor autocommando
PTRFIL	F864–F865	beginadres FCB van de actuele file bij file I/O
CLOC	F92A	adres in VRAM van pixelpatroon, overeenkomend met actuele schermpositie
CMASK	F92C	masker voor byte uit videogeheugen dat wordt aangeduid door CLOC
DPPAGE	FAF5	zichtbaar scherm
ACPAGE	FAF6	actief scherm
EXBRSA	FAF8	slotadres op SUB-ROM
LOGOPR	FB02	code voor logische bewerking (lage nybble)
	FB21	informations of disk drivers
VOICEN	FB38	werkruimte voor PLAY-statement
QUEUEN	FB3E	nummer van de geluidsqueue
FNKFLG	FBCE–FBD7	vlag voor interrupt faciliteit per funktietoets
PATWRK	FC40–FC47	bewaarplaats voor patroon van ASCII-teken
HIMEM	FC4A	beginadres van BASIC werkgebied
SCRMOD	FCAF	schermmodus
GXPOS	FCB3–FCB4	laatst bereikte X-positie op grafisch scherm
GYPOS	FCB5–FCB6	laatst bereikte Y-positie op grafisch scherm
EXPTBL	FCC1	slotadres van MAIN-ROM
H-PHYD	FFA7	disk is aangesloten als de inhoud van dit adres niet gelijk is aan '&HC9'
H-STKE	FEDA	autostart RAM-haak
	FFFF	slot select register voor het secundaire slot

Adres	*Naam*	*Functie*
0000	CHKRAM	RAM-controle en -selectie voor BASIC EN OS. Gewijzigde registers: Alle registers
0004	CGTABL	Pointer naar tekenpatroontabel.
0006	VDP.DR	Pointer naar poortadres om data uit VRAM te lezen.
0007	VDP.DW	Pointer naar poortadres om data naar VRAM te schrijven.
0008	SYNCHR	Syntax-controle in BASIC-tekst. Resultaat is syntax error of sprong naar CHRGTR. Invoerparameters: HL=pointer naar volgend BASIC-teken Uitvoerparameters: HL=bijgewerkte pointer; A=bevat teken; C-vlag=1 bij getal; Z-vlag=1 bij einde statement Gewijzigde registers: AF, HL
000C	RDSLT	Selectie van slot en lezen van geheugeninhoud. Invoerparameters: A=FxxxSSPP PP — primair slot (0–3) SS — secundair slot (0–3) F — F=1 bij sec. slot Uitvoerparameters: A bevat geheugeninhoud van adres Gewijzigde registers: AF,BC,DE
0010	CHRGTR	Lezen van volgende teken of token uit BASIC-tekst. Zie SYNCHR.
0014	WRSLT	Selectie van slot en schrijven van waarde in geheugen. Zie RDSLT. Invoerparameters: E=waarde Gewijzigde registers: AF,BC,D
0018	OUTDO	Uitvoer van teken naar actueel apparaat. Invoerparameters: A=ASCII-code teken; PTRFIL, PRTFLG
001C	CALSLT	Aanroepen subroutine in willekeurig slot. Invoerparameters: IYH=FxxxSSPP (voor betekenis zie RDSLT) IX bevat adres
0020	DCOMPR	Vergelijking van HL met DE. Invoerparameters: HL, DE Uitvoerparameters: vlaggen Gewijzigde registers: AF
0024	ENASLT	Selectie en permanente activering van slot. Zie RDSLT. Gewijzigde registers: alle
0028	GETYPR	Bepalen van type van laatst geëvalueerde expressie. Invoerparameters: VALTYP Uitvoerparameters: vlaggen Gewijzigde registers: AF
002B		ID-bytes
002D		Gereserveerd voor toekomstige uitbreiding (002D-002F). In deze adressen staan nullen.
0030	CALLF	Aanroepen subroutine in willekeurig slot.
0038	KEYINT	Uitvoeren van hardware interrupts.
003B	INITIO	Initialiseren van I/O-apparaten. Gewijzigde registers: Alle

Adres	*Naam*	*Functie*
003E	INIFN	Initialiseren van definities functietoetsen. Zie INITIO.
0041	DISSCR	Uitschakelen van scherm-display. Gewijzigde registers: AF, BC
0044	ENASCR	Inschakelen van scherm-display. Zie DISSCR.
0047	WRTVDP	Schrijven naar VDP-register. Invoerparameters: C=register; B=data Gewijzigde registers: AF, BC
004A	RDVRM	Lezen van VRAM-adres. Invoerparameters: HL=adres Uitvoerparameters: A=inhoud Gewijzigde registers: AF
004D	WRTVRM	Schrijven naar VRAM-adres. Invoerparameters: HL=adres; A=data Gewijzigde registers: AF
0050	SETRD	Initialiseert VDP voor leesoperatie. Wordt aangeroepen door RDVRM en LDIRMV. Invoerparameters: HL=waarde Gewijzigde registers: AF
0053	SETWRT	Initialiseert VDP voor schrijfoperatie. Wordt aangeroepen door WRTVRM, FILVRM en LDIRVM. Zie verder SETRD.
0056	FILVRM	Vullen van VRAM met opgegeven data. Invoerparameters: HL=adres; BC=lengte; A=data Gewijzigde registers: AF, BC
0059	LDIRMV	Kopiëren geheugenblok van VRAM naar gewone geheugen. Invoerparameters: HL=bronadres; DE=doeladres; BC=lengte Gewijzigde registers: Alle
005C	LDIRVM	Kopiëren geheugenblok van gewone geheugen naar VRAM. Zie LDIRMV.
005F	CHGMOD	Initialiseren van het scherm. Invoerparameters: A=SCRMOD (0-8) Gewijzigde registers: Alle
0062	CHGCLR	Veranderen van schermkleuren. Invoerparameters: A=scherm-mode; FORCLR=voorgrondkleur; BAKCLR=achtergrondkleur; BDRCLR=schermrand Gewijzigde registers: Alle
0066	NMI	Uitvoeren van non-maskable interrupts.
0069	CLRSPR	Initialiseren van alle sprites. Invoerparameters: SCRMOD Gewijzigde registers: Alle
006C	INITXT	Initialiseren van scherm voor tekstmode (40 × 24) en instellen VDP. Invoerparameters: TXTNAM,TXTCGP Gewijzigde registers: Alle
006F	INIT32	Als INITXT, nu voor (32 × 24) scherm. Invoerparameters: T32NAM,T32CGP,T32COL,T32ATR,T32PAT Gewijzigde registers: Alle

Adres	*Naam*	*Functie*
0072	INIGRP	Als INITXT, nu voor hires-mode. Invoerparameters: GRPNAM,GRPCGP,GRPCOL,GRPATR, GRPPAT Gewijzigde registers: Alle
0075	INIMLT	Als INITXT, nu voor multicolor-mode. Invoerparameters: MLTNAM,MLTCGP,MLTCOL,MLTATR, MLTPAT Gewijzigde registers: Alle
0078	SETTXT	Instellen van VDP voor tekstmode (40 × 24). Zie INITXT.
007B	SETT32	Als SETTXT, nu voor (32 × 24) scherm. Zie INIT32.
007E	SETGRP	Als SETTXT, nu voor hires-mode. Zie INIGRP.
0081	SETMLT	Als SETTXT, nu voor multicolor-mode. Zie INIMLT.
0084	CALPAT	Berekenen van beginadres van sprite-patroontabel. Invoerparameters: A=sprite ID (nummer) Uitvoerparameters: HL=adres Gewijzigde registers: AF,DE,HL
0087	CALATR	Berekenen van beginadres van sprite-attribuuttabel Invoerparameters: A=sprite ID Uitvoerparameters: HL=adres Gewijzigde registers: AF, DE, HL
008A	GSPSIZ	Opvragen van de actuele sprite-grootte. Uitvoerparameters: A=grootte in bytes; C-vlag=1 bij 16x.16 sprite, C-vlag=0 bij 8x.8 sprite. Gewijzigde registers: AF
008D	GRPPRT	Plaatsen van teken op grafisch scherm. Invoerparameters: A=code van teken; LOGOPR (bij scherm 5–8) bevat code logische operatie.
0090	GICINI	Initialiseren van geluidssysteem. (PSG: Programmable Sound Generator). Gewijzigde registers: Alle
0093	WRTPSG	Schrijven van waarde naar PSG-register. Invoerparameters: A=register; E=data
0096	RDPSG	Lezen van inhoud van PSG-register. Invoerparameters: A=register Uitvoerparameters: A=data
0099	STRTMS	Controle c.q. start van geluidssysteem. Gewijzigde registers: Alle
009C	CHSNS	Controle van status van keyboard-buffer. Uitvoerparameters: Z-vlag=1 als keyboard-buffer leeg. Gewijzigde registers: AF
009F	CHGET	Wachten op en binnenhalen van invoer toetsenbord. Uitvoerparameters: A=code teken Gewijzigde registers: AF
00A2	CHPUT	Sturen van teken naar scherm. Invoerparameters: A=code teken

Adres	Naam	Functie
00A5	LPTOUT	Sturen van teken naar printer. Invoerparameters: A=code teken Uitvoerparameters: C-vlag=1 als uitvoer wordt afgebroken. Gewijzigde registers:F
00A8	LPTSTT	Controle van status printer. Uitvoerparameters: A=255 en Z-vlag=0 als printer gereed is, anders A=0 en Z-vlag=1 Gewijzigde registers:AF
00AB	CNVCHR	Controle van graphic header byte en convertering code. Invoerparameters: A=code teken Uitvoerparameters: C-vlag=0:graphic header C-vlag=1, Z-vlag=1:geconverteerde code in A C-vlag=1,Z-vlag=0:niet geconverteerde code Gewijzigde registers:AF
00AE	PINLIN	Invoer van regel van keyboard vanaf kolom 1 (of andere bij AUTO) tot CR of CTRL-STOP. Uitvoerparameters: HL=adres buffertop–1 C-vlag=1 bij CTRL-STOP Gewijzigde registers:Alle
00B1	INLIN	Als PINLIN echter vanaf willekeurige kolom. Zie PINLIN.
00B4	QINLIN	Uitvoer van "? ", vervolgens INLIN. Zie PINLIN.
00B7	BREAKX	Controle van status CTRL-STOP toets. Uitvoerparameters: C-vlag=1 als toets is ingedrukt Gewijzigde registers:AF
00BA	ISCNTC	Controle van status van SHIFT-STOP toets. BASIC keert terug naar commando-mode bij toetsindruk
00BD	CKCNTC	Als ISCNTC. De routine wordt gebruikt door de BASIC interpreter. Zie ISCNTC.
00C0	BEEP	Opwekken van een "beep"-toon. Gewijzigde registers:Alle
00C3	CLS	Wissen van scherm. Invoerparameters: Z-vlag=1. Als Z-vlag=0 dan is deze BIOS-call onwerkzaam. Gewijzigde registers:AF,BC,DE
00C6	POSIT	Positioneren van cursor. Invoerparameters: H=kolom; L=regel Gewijzigde registers:AF
00C9	FNKSB	Opnieuw weergeven van functietoets-display. Invoerparameters: CNSDFG>0:functietoetsen tonen (aanroep van DSPFNK). CNSDFG=0:niet tonen Gewijzigde registers:Alle
00CC	ERAFNK	Uitschakelen van functietoets-display. Gewijzigde registers:Alle
00CF	DSPFNK	Inschakelen van functietoets-display. Gewijzigde registers:Alle
00D2	TOTEXT	Zetten van scherm in tekstmode. Gewijzigde registers:Alle

Adres	*Naam*	*Functie*
00D5	GTSTCK	Opvragen van status joystick.
		Invoerparameters: A=joystick ID
		Uitvoerparameters: A=richting
		Gewijzigde registers: Alle
00D8	GTTRIG	Opvragen van status vuurknop.
		Invoerparameters: A=vuurknop ID
		Uitvoerparameters: A=255:ingedrukt
		A=0:niet ingedrukt
		Gewijzigde registers: Alle
00DB	GTPAD	Opvragen status touch pad.
		Invoerparameters: A=ID
		Uitvoerparameters: A=waarde
		Gewijzigde registers: Alle
00DE	GTPDL	Opvragen van status paddle.
		Invoerparameters: A=paddle ID
		Uitvoerparameters: A=paddle stand
		Gewijzigde registers: Alle
00E1	TAPION	Aanzetten van motor en lezen van header van tape.
		Uitvoerparameters: C-vlag=1 bij onderbreking
		Gewijzigde registers: Alle
00E4	TAPIN	Lezen van data van tape.
		Uitvoerparameters: A=data; C-vlag=1 bij onderbreking
		Gewijzigde registers: Alle
00E7	TAPIOF	stoppen van lezen van tape.
00EA	TAPOON	Aanzetten van motor en schrijven van header naar tape.
		Invoerparameters: A=0:korte header
		A<>0:lange header
		Uitvoerparameters: C-vlag=1 bij onderbreking
		Gewijzigde registers: Alle
00ED	TAPOUT	Schrijven van data naar tape.
		Invoerparameters: A=data
		Uitvoerparameters: C-vlag=1 bij onderbreking
		Gewijzigde registers: Alle
00F0	TAPOOF	Stoppen van schrijven naar tape.
		Uitvoerparameters: C-vlag=1 bij onderbreking
00F3	STMOTR	Starten/stoppen van cassettemotor.
		Invoerparameters: A=0:stopt motor
		A=1:start motor
		A=255:wisseling motorstatus
		Gewijzigde registers: AF
00F6	LFTQ	Opvragen van aantal overgebleven bytes in queue.
		Invoerparameters: A=nummer queue
		Uitvoerparameters: A=aantal bytes
		Gewijzigde registers: AF,BC,HL
00F9	PUTQ	Zetten van een byte in queue.
		Invoerparameters: A=nummer queue
		E=waarde

Adres	Naam	Functie
		Uitvoerparameters: Z-vlag=1:queue vol
		Gewijzigde registers:AF,BC,HL
00FC	RIGHTC	Bewegen van één pixel naar rechts.
		Gewijzigde registers:AF
00FF	LEFTC	Bewegen van één pixel naar links.
		Gewijzigde registers:AF
0102	UPC	Bewegen van één pixel naar boven.
		Gewijzigde registers:AF
0105	TUPC	Zie UPC.
0108	DOWNC	Bewegen van één pixel naar beneden.
		Gewijzigde registers:AF
010B	TDOWNC	ZIE DOWNC.
010E	SCALXY	Controle of punt binnen scherm valt; bij negatief resultaat wordt omgeschaald.
		Invoerparameters: BC=x-positie
		DE=y-positie
		Uitvoerparameters: BC="nieuwe" x-positie
		DE="nieuwe" y-positie
		Gewijzigde registers:AF
0111	MAPXYC	Omzetten coördinaten in absoluut adres.
		Invoerparameters: BC=x-positie
		DE=y-positie
		Uitvoerparameters: SCREEN 2–4:HL=adres; A=masker
		SCREEN 5–8:HL=x-positie; A=y-positie
		Gewijzigde registers:F
0114	FETCHC	Opvragen van actueel adres en masker.
		Uitvoerparameters: HL=adres; A=masker
		Gewijzigde registers:F
0117	STOREC	Opslaan van adres en masker.
		Invoerparameters: HL=adres; A=masker
011A	SETATR	Veranderen van werkkleur voor grafische acties.
		Invoerparameters: A=werkkleur
		Uitvoerparameters: C-vlag=1 bij ongeoorloofde code
		Gewijzigde registers:F
011D	READC	Bepalen van kleur van actuele pixel.
		Uitvoerparameters: A=kleurcode
		Gewijzigde registers:F
0120	SETC	Veranderen van kleur van actuele pixel.
		Gewijzigde registers:AF
0123	NSETCX	Zetten van pixels in horizontale richting.
		Invoerparameters: HL=teller
		Gewijzigde registers:Alle
0126	GTASPC	Opvragen van aangezichtsverhouding.
		Uitvoerparameters: DE,HL
0129	PNTINI	Initialiseren van paint.
		Gewijzigde registers:AF
012C	SCANR	Scannen van pixels naar rechts bij PAINT.

Adres	Naam	Functie
		Invoerparameters: B=pseudo vlag
		DE=randteller
		Uitvoerparameters: C=veranderde vlag
		DE=randteller
		Gewijzigde registers: Alle
012F	SCANL	Scannen van pixels naar links bij PAINT. Zie SCANR.
0132	CHGCAP	Veranderen van status van CAPS-lock lamp.
		Invoerparameters: A=0:lamp uit
		A<>0:lamp aan
		Gewijzigde registers: AF
0135	CHGSND	Veranderen van status 1-bit geluidspoort.
		Invoerparameters: A=0:uitschakelen
		A<>0:inschakelen
		Gewijzigde registers: AF
0138	RSLREG	Lezen van actuele waarde van primair slot-register.
		Uitvoerparameters: A=resultaat
013B	WSLREG	Schrijven naar primair slot-register.
		Invoerparameters: A=data
013E	RDVDP	Lezen van status register van VDP.
		Uitvoerparameters: A=data
0141	SNSMAT	Opvragen van status van rij uit keyboard matrix.
		Invoerparameters: A=rij
		Uitvoerparameters: A=status. Overeenkomstig bit wordt 0 bij contact
		Gewijzigde registers: AF,C
0144	PHYDIO	Uitvoeren van bewerking voor massageheugens (b.v. disk).
0147	FORMAT	Initialiseren van massageheugen.
014A	ISFLIO	Controle of I/O gaande is.
		Uitvoerparameters: Vlaggen
		Gewijzigde registers: AF
014D	OUTDLP	Uitvoer van teken naar printer.
		Invoerparameters: A=code teken
		Gewijzigde registers: F
0150	GETVCP	Opvragen van pointer naar muziekqueue.
		Invoerparameters: A=kanaalnummer
		Uitvoerparameters: HL=pointer
		Gewijzigde registers: AF
0153	GETVC2	Bepalen van adres van veld binnen het VCB.
		Invoerparameters: L=gewenste verschuiving in stembuffer
		QUEUEN=nummer van de stem
		Uitvoerparameters: HL=pointer
		Gewijzigde registers: AF
0156	KILBUF	Wissen van keyboard buffer.
		Gewijzigde registers: HL
0159	CALBAS	Aanroepen van BASIC-interpreter.
		Invoerparameters: IX=adres
015C	SUBROM	Aanroepen van subroutine in willekeurig slot van SUBROM.

Adres	Naam	Functie
		Invoerparameters: IX op stapel. Adres in IX
		Gewijzigde registers: Alternatieve registers, IY
015F	EXTROM	Aanroepen van subroutine in willekeurig slot van extended ROM
		Invoerparameters: Adres in IX
		Gewijzigde registers: Zie SUBROM
0162	CHKSLZ	Slotscan voor SUBROM.
		Gewijzigde registers: Alle
0165	CHKNEW	Controle van schermmode.
		Uitvoerparameters: C-vlag=0 bij SCREEN 5–8
		Gewijzigde registers: AF
0168	EOL	Wissen tot einde van regel.
		Invoerparameters: H=kolom; L=regel
		Gewijzigde registers: Alle
016B	BIGFIL	Vullen van VRAM met opgegeven waarde. Zie ook FILVRM.
		Invoerparameters: HL=adres; BC=lengte; A=data
		Gewijzigde registers: AF,BC
016E	NSETRD	Instellen VDP om te lezen.
		Invoerparameters: HL=adres
		Gewijzigde registers: AF
0171	NSTWRT	Instellen VDP om te schrijven.
		Invoerparameters: HL=adres
		Gewijzigde registers: AF
0174	NRDVRM	Lezen van VRAM.
		Invoerparameters: HL=adres
		Uitvoerparameters: A=data
		Gewijzigde registers: F
0177	NWRVRM	Schrijven naar VRAM.
		Invoerparameters: HL=adres; A=data
		Gewijzigde registers: AF

EXTENDED ROM

Adres	Naam	Functie
0069	PAINT	Inkleuren van grafisch scherm(mode 5–8).
		Invoerparameters: HL=tekstpointer naar BASIC-token
		Uitvoerparameters: HL=bijgewerkte pointer
		Gewijzigde registers: Alle
006D	PSET	Zetten van een punt(mode 5–8). Zie PAINT.
0071	ATRSCN	Bepalen van de werkkleur(mode 5–8). Zie PAINT.
0075	GLINE	Tekenen van een lijn(mode 5–8). Zie PAINT.
0079	DOBOXF	Tekenen van een ingekleurde rechthoek(mode 5–8). Zie PAINT.
		Invoerparameters: (BE,DE)=beginpunt (GXPOS,GYPOS)=eindpunt
007D	DOLINE	Tekenen van een lijn(mode 5–8). Zie DOBOXF.
0081	BOXLIN	Tekenen van een rechthoek(mode 5–8). Zie DOLINE.
0085	DOGRPH	Tekenen van een lijn(mode 5–8).
		Invoerparameters: coördinaten als DOBOXF; ATRBYT bevat kleurcode; LOGOPR code logische operatie
		Gewijzigde registers: AF

Adres	*Naam*	*Functie*
0089	GRPPRT	Plaatsen van teken op scherm(mode 5–8). Invoerparameters: A=code van teken; ATRBYT bevat kleurcode; LOGOPR bevat code logische operatie.
008D	SCALXY	Controle of punt binnen scherm valt; bij negatief resultaat wordt omgeschaald. Invoerparameters: BC=x-positie DE=y-positie Uitvoerparameters: BC="nieuwe" x-positie DE="nieuwe" y-positie Gewijzigde registers: AF
0091	MAPXYC	Omzetten van coördinaten in absoluut adres (SCREEN 3 en 5–8). Invoerparameters: BC=x-positie DE=y-positie Uitvoerparameters: SCREEN 3: Adres=HL en CLOC; Bitmasker=A en CMASK. SCREEN 5–8: X-positie=HL en CLOC; Y-positie=A en CMASK. Gewijzigde registers: F
0095	READC	Bepalen van kleur van actuele pixel (SCREEN 3 en 5–8). Invoerparameters: Coördinaten in CLOC en CMASK. Uitvoerparameters: A=kleurcode Gewijzigde registers: AF
0099	SETATR	Veranderen van werkkleur voor grafische acties. Invoerparameters: A=kleurcode Uitvoerparameters: C-vlag=1 bij ongeoorloofde waarde. Gewijzigde registers: F
009D	SETC	Veranderen van kleur van actuele pixel (SCREEN 3 en 5–8). Invoerparameters: coördinaten in CLOC en CMASK; ATRBYT bevat kleurcode Gewijzigde registers: AF
00A1	TRIGHT	Bewegen van één pixel naar rechts (SCREEN 3 en 5–8). Invoerparameters: Zie READC Uitvoerparameters: Nieuwe coördinaat: CLOC en CMASK C-vlag=1 bij bereiken schermrand Gewijzigde registers: AF
00A5	RIGHTC	Als TRIGHT doch alleen voor SCREEN 3. Invoerparameters: Zie TRIGHT Uitvoerparameters: Nieuwe coördinaat: CLOC en CMASK Gewijzigde registers: AF
00A9	TLEFTC	Bewegen van één pixel naar links (SCREEN 3 en 5–8). Zie TRIGHT.
00AD	LEFTC	Zie TLEFTC doch alleen voor SCREEN 3. Zie RIGHTC.
00B1	TDOWNC	Bewegen van één pixel naar beneden (SCREEN 3 en 5–8). Zie TRIGHT.
00B5	DOWNC	Als TDOWNC doch alleen voor SCREEN 3. Zie PIGHTC.
00B9	TUPC	Bewegen van één pixel naar boven (SCREEN 3 en 5–8). Zie TRIGHT.

Adres	Naam	Functie
00BD	UPC	Als TUPC doch alleen voor SCREEN 3. Zie RIGHTC.
00C1	SCANR	Scannen van pixels naar rechts (PAINT) (SCREEN 3 en 5–8). Invoerparameters: B=pseudo vlag DE=randteller Uitvoerparameters: C=veranderde vlag DE=randteller Gewijzigde registers: Alle
00C5	SCANL	Scannen van pixels naar links (PAINT) (SCREEN 3 en 5–8). Zie SCANR.
00C9	NVBXLN	Tekenen van een rechthoek (SCREEN 5–8). Invoerparameters: BC,DE=beginpunt GXPOS,GYPOS=eindpunt ATRBYT bevat kleurcode; LOGOPR bevat code logische operatie. Gewijzigde registers: Alle
00CD	NVBXFL	Tekenen van een ingekleurde rechthoek (SCREEN 5–8). Zie NVBXLN.
00D1	CHGMOD	Instellen van VDP. Invoerparameters: A=SCRMOD Gewijzigde registers: Alle
00D5	INITXT	Initialiseren van scherm, tekstmode (40×24) en instellen van VDP Invoerparameters: TXTNAM,TXTCGP Gewijzigde registers: Alle
00D9	INIT32	Als INITXT, nu voor (32×24)-scherm. Invoerparameters: T32NAM,T32CGP,T32COL,T32ATR,T32PAT Gewijzigde registers: Alle
00DD	INIGRP	Alle INITXT, nu voor hires-mode. Invoerparameters: GRPNAM,GRPCGP,GRPCOL,GRPATR, GRPPAT Gewijzigde registers: Alle
00E1	INIMLT	Als INITXT, nu voor multicolor-mode. Invoerparameters: MLTNAM,MLTCGP,MLTCOL,MLTATR, MLTPAT Gewijzigde registers: Alle
00E5	SETTXT	Instellen van VDP voor tekstmode (40×24). Invoerparameters: TXTNAM,TXTCGP Gewijzigde registers: Alle
00E9	SETT32	Als SETTXT, nu voor (32×24)-scherm. Zie INIT32.
00ED	SETGRP	Als SETTXT, nu voor hires-mode. Zie INIGRP.
00F1	SETMLT	Als SETTXT, nu voor multicolor-mode. Zie INIMLT.
00F5	CLRSPR	Initialiseren van alle sprites. Invoerparameters: SCRMOD Gewijzigde registers: Alle
00F9	CALPAT	Berekenen van beginadres van spritepatroontabel. Equivalent met MSX1-BIOS-call

Adres	*Naam*	*Functie*
		Invoerparameters: A=sprite ID
		Uitvoerparameters: HL=adres
		Gewijzigde registers: AF,DE,HL
00FD	CALATR	Berekenen van beginadres van sprite-attribuuttabel. Zie CALPAT. Equivalent met MSX-1-BIOS-call
0101	GSPSIZ	Opvragen van actuele sprite-grootte. Equivalent met MSX1-BIOS-call.
		Uitvoerparameters: A=grootte in bytes C-vlag=1 bij 16×16 sprite; C-vlag=0 bij 8×8 sprite.
		Gewijzigde registers: AF
0105	GETPAT	Opvragen van tekenpatroon. Equivalent met MSX1-BIOS-call.
		Invoerparameters: A=ASCII-code van teken
		Uitvoerparameters: PATWRK=patroon
		Gewijzigde registers: Alle
0109	WRTVRM	Schrijven naar VRAM-adres.
		Invoerparameters: HL=data A=data
		Gewijzigde registers: AF
010D	RDVRM	Lezen van VRAM-adres.
		Invoerparameters: HL=adres
		Uitvoerparameters: A=data
		Gewijzigde registers: AF
0111	CHGCLR	Veranderen van schermkleuren.
		Invoerparameters: A=schermmode FORCLR=voorgrondkleur BAKCLR=achtergrondkleur BDRCLR=schermrandkleur
		Gewijzigde registers: Alle
0115	CLS	Wissen van scherm.
		Gewijzigde registers: Alle
0119	CLRTXT	Wissen van tekstscherm.
		Gewijzigde registers: Alle
011D	DSPFNK	Inschakelen van funktietoets-display.
		Gewijzigde registers: Alle
0121	DELLNO	Wissen van een regel in tekstmode.
		Invoerparameters: L=regelnummer
		Gewijzigde registers: Alle
0125	INSLNO	Invoegen van een regel in tekstmode.
		Invoerparameters: L=regelnummer
		Gewijzigde registers: Alle
0129	PUTVRM	Plaatsen van teken op tekstscherm.
		Invoerparameters: H=kolomnummer L=regelnummer
		Gewijzigde registers: AF
012D	WRTVDP	Schrijven naar VDP-register.

Adres	Naam	Functie
		Invoerparameters: C=register B=data Gewijzigde registers:AF,BC
0131	VDPSTA	Lezen van statusregister van VDP. Invoerparameters: A=statusregister (0-9) Uitvoerparameters: A=data Gewijzigde registers:F
0135	KYKLOK	Behandelen van kana-toets* en lamp. Gewijzigde registers:AF
0139	PUTCHR	Omzetten van ingetoetst teken van kana* en dit in een buffer stoppen. Invoerparameters: Z-vlag=1 als geen conversie dient plaats te vinden Gewijzigde registers:Alle
013D	SETPAG	Instellen van VDP voor paginaveranderingen. Invoerparameters: ACPAGE,DPPAGE Gewijzigde registers:AF
0141	INIPLT	Initialiseren van palet en VRAM. Gewijzigde registers:AF,BC,DE
0145	RSTPLT	Terugzetten van palet vanuit VRAM. Gewijzigde registers:AF,BC,DE
0149	GETPLT	Opvragen van kleurcodes van palet. Invoerparameters: A=palet (0–15) Uitvoerparameters: Hoge nybble B=rood Lage nybble B=blauw Lage nybble C=groen Gewijzigde registers:AF,DE
014D	SETPLT	Instellen van palet kleurcodes. Invoerparameters: D=palet(0–15) Hoge nybble A=rood Lage nybble A=blauw Lage nybble E=groen Gewijzigde registers:AF
0151	PUTSPR	Plaatsen van sprites. Invoerparameters: HL=tekstpointer naar BASIC Uitvoerparameters: HL=bijgewerkte pointer Gewijzigde registers:Alle
0155	COLOR	Veranderen van scherm-, sprite- en paletkleur. Zie PUTSPR.
0159	SCREEN	Veranderen van schermmode. Zie PUTSPR.
015D	WIDTHS	Veranderen van tekstbreedte. Zie PUTSPR.
0161	VDP	Instellen van VDP-register. Zie PUTSPR.
0165	VDPF	Lezen van actueel VDP-register. Zie PUTSPR.
0169	BASE	Instellen van VDP-basisregisters. Zie PUTSPR.
016D	BASEF	Lezen van VDP-basisregisters. Zie PUTSPR.
0171	VPOKE	Schrijven van byte naar VRAM. Zie PUTSPR.

* kana is een uit het Japans (oorsprong van het MSX-concept) afkomstige term. Op de meeste MSX-computers is de kana-toets de z.g. code-toets, die letters met accenten genereert.

Adres	*Naam*	*Functie*
0175	VPEEK	Lezen van byte van VRAM. Zie PUTSPR.
0179	SETS	Instellen van beeptoon, scherm, tijd en datum. Zie PUTSPR.
017D	BEEP	Opwekken van beeptoon.
		Gewijzigde registers: Alle
0181	PROMPT	Tonen van prompt.
		Gewijzigde registers: Alle
0185	SDFSCR	Terugzetten van schermparameters vanuit RAM (klokchip).
		Invoerparameters: C-vlag=1 bij aanroepen vanuit DOS
		Gewijzigde registers: Alle
0189	SETSCR	Terugzetten van scherm en afdrukken van beginboodschap.
		Gewijzigde registers: Alle
018D	SCOPY	Kopiëren van VRAM, array en diskfile
		Invoerparameters: HL=tekstpointer
		Uitvoerparameters: HL=bijgewerkte tekstpointer
		Gewijzigde registers: Alle
0191	BLTVV	Kopiëren van VRAM naar VRAM.
		Invoerparameters: HL=0F562H
		Gewijzigde registers: Alle
0195	BLTVM	Kopiëren van array naar VRAM. Zie BLTVV.
0199	BLTMV	Kopiëren van VRAM naar array. Zie BLTVV.
019D	BLTVD	Kopiëren van disk-file naar VRAM. Zie BLTVV.
01A1	BLTDV	Kopiëren van VRAM naar disk-file. Zie BLTVV.
01A5	BLTMD	Laden van array van disk-file. Zie BLTVV.
01A9	BLTDM	Opslaan van array in disk-file. Zie BLTVV.
01AD	NEWPAD	Inlezen van status van paddle, muis en trackerball.
		Invoerparameters: A=8 controleer lichtpen (255=aanwezig)
		A=9
		geef X-positie
		A=10
		geef Y-positie
		A=11
		geef status lichtpen-knop (255=ingedrukt)
		A=12
		controleer muis op poort 1 (255=aanwezig)
		A=13
		geef X-offset
		A=14
		geef Y-offset
		A=16
		controleer muis op poort 2 (255=aanwezig)
		A=17
		geef X-offset
		A=18
		geef Y-offset
		Uitvoerparameters: A=status
		Gewijzigde registers: Alle
01B1	GETPUT	Opvragen van tijd en datum en plaatsen van kanji.

Adres	Naam	Functie
		Invoerparameters: HL=tekstpointer naar BASIC
		Uitvoerparameters: HL=bijgewerkte pointer
		Gewijzigde registers:Alle
01B5	CHGMDP	Instellen van VDP en initialiseren van palet.
		Invoerparameters: A=schermmode (0–8)
		Gewijzigde registers:Alle
01B9	RESV1	Gereserveerd voor toekomstig gebruik.
01BD	KNJPRT	Plaatsen van kanji*-teken op grafisch scherm (mode 5–8).
		Invoerparameters: BC=JIS*-code kanji*-teken
		A=display mode (0–2)
		Gewijzigde registers:AF
01F5	REDCLK	Lezen van de interne klok.
		Invoerparameters: C=RAM adres klok
		Uitvoerparameters: A=gelezen data
		Gewijzigde registers:F
01F9	WRTCLK	Zetten van de interne klok
		Invoerparameters: C=RAM adres klok
		A=te schrijven data
		Gewijzigde registers:F

* Evenals kana is kanji een Japanse term. Een kanji-teken wordt via het toetsenbord met de kana-toets (code-toets) opgeroepen. De JIS-code is een Japanse standaard.

OVERZICHT MSX-BASIC-OPDRACHTEN

In dit overzicht worden alle BASIC-instructies, -commando's, systeemvariabelen en functies kort behandeld.
Van iedere instructie en van ieder commando wordt de volledige syntax (dat wil zeggen de volledige schrijfwijze) gegeven.
Daarbij worden twee symbolen gebruikt met de volgende betekenis:

[...] alles tussen vierkante haken is optioneel, dat wil zeggen: kan eventueel worden weggelaten;

< ... > gegevens tussen deze haken moeten door de gebruiker zelf worden ingevuld.

In een aantal uitdrukkingen treft men 3 punten aan bijvoorbeeld bij:

CALL <naam> [<string>,<string>...]

De drie punten aan het eind van deze uitdrukking willen zeggen 'de uitdrukking kan op de getoonde wijze worden voortgezet'. In dit geval kunnen er dus meer strings, gescheiden door een komma worden opgenomen.

Dit overzicht geeft tevens de volledige syntax van alle BASIC-functies en systeemvariabelen.
Daarbij worden twee notaties gebruikt met de volgende betekenis:

<X> een numerieke variabele of uitdrukking die de gebruiker zelf moet invullen;

<X$> een string-variabele of uitdrukking die de gebruiker zelf moet invullen.

Bij elke opdracht wordt de categorie vermeld: instructie, commando, systeemvariabele of functie.
Na het uitvoeren van een commando zal MSX-BASIC de 'prompt' Ok op het beeldscherm afbeelden en wachten totdat het volgende commando wordt ingetoetst.
Een functie of systeemvariabele kan alleen tezamen met een instructie of commando worden gebruikt.

ABS(<X>)
Geeft de absolute waarde van <X>.
Categorie: functie
Voorbeeld: `PRINT ABS(-3)`

ASC(<X$>)
Geeft de ASCII-code van het eerste schriftteken van <X$>.
Categorie: functie
Voorbeeld: `PRINT ASC("MSX")`

ATN(<X>)
Geeft de arctangens van <X> in radialen. Het resultaat ligt tussen $-\pi/2$ en $\pi/2$.
Categorie: functie
Voorbeeld: `PRINT ATN(3)`

AUTO [<regelnummers>][,[<stapgrootte>]]
Genereert automatisch regelnummers tijdens het invoeren van een programma. Als geen <regelnummer> wordt opgegeven, dan wordt begonnen met regelnummer 10 en anders met het opgegeven <regelnummer>. Is geen <stapgrootte> opgegeven dan wordt het regelnummer steeds met 10 verhoogd, anders met de opgegeven <stapgrootte>.
Als automatisch een regelnummer wordt gegenereerd dat reeds gebruikt was ziet men als waarschuwing een asterisk *.
Het commando wordt beëindigd met CTRL/C, dat wil zeggen met ingedrukte CTRL-toets wordt op C gedrukt.
Categorie: commando
Voorbeeld: `AUTO 100,50`

BASE (<X>)
Bevat het eerste adres van de zogenaamde Video Display Processor (VDP-tabellen)
De volgende tabel geeft een overzicht van de gebruikte afkortingen.

tm	= tekst-modus
gm	= grafische modus
pt	= patroontabel
spt	= sprite-patroontabel
spat	= sprite-attribuuttabel
ct	= color (kleur)-tabel
nt	= naamtabel

<X> Betekenis	*<X> Betekenis*	*<X> Betekenis*	*<X> Betekenis*
0 nt in tm1	13 spat in gm1	24 spt in gm3	34 spt in gm5
2 pt in tm1	14 spt in gm1	25 nt in gm4	35 nt in gm6
5 nt in tm2	15 nt in gm2	26 ct in gm4	36 ct in gm6
6 ct in tm2	16 pt in gm2	27 pt in gm4	37 pt in gm6
7 pt in tm2	17 spat in gm2	28 spat in gm4	38 spat in gm6
8 spat in tm2	19 spt in gm2	29 spt in gm4	39 spt in gm6
9 spt in tm2	20 nt in gm3	30 nt in gm5	40 nt in gm7
10 nt in gm1	21 ct in gm3	31 ct in gm5	41 ct in gm7
11 ct in gm1	22 pt in gm3	32 pt in gm5	42 pt in gm7
12 pt in gm1	23 spat in gm3	33 spat in gm5	43 spat in gm7
			44 spt in gm7

In BASE-adres 0 t/m 19 kan alleen een waarde worden geschreven. Dit houdt in dat als men in SCREEN modus 2 of 3 een BASE-adres verandert, dit ook geldt voor de SCREEN modus 4 t/m 8.
Categorie: systeemvariabele
Voorbeeld:
```
10 SCREEN 0
20 PRINT BASE(2):END
```

BEEP
Genereert een pieptoon met een beperkte duur.
Hetzelfde effect kan men met PRINT CHR$(7) bereiken.
Categorie: instructie
Voorbeeld:
```
10 FOR K=1 TO 1000:PRINT K:NEXT
20 BEEP:END
```

BIN$ (<X>)
Zet het aangegeven getal om in een binaire notatie.
<X> moet liggen tussen −32768 en 65535. Als<X> negatief is, krijgt men de zogenaamde 'two's complement' vorm.
Categorie: functie
Voorbeeld: `PRINT BIN$(100)`

BLOAD "<dev>:[<filenaam>]"[,R][,<verschuiving>]
Hiermee wordt een in machinecode geschreven programma geladen dat met een BSAVE-commando is opgeslagen.
Vult men voor <dev> de term CAS in dan wordt daarmee de cassetterecorder aange geven.
Men kan voor<dev> ook de letters A t/m F invullen. Deze hebben dan betrekking op de diskdrives 1 t/m 6.
De term <filenaam> slaat op de naam die is meegegeven (max. 6 lettertekens voor de cassetterecorder en 8 voor de diskdrive). Geeft men de R op dan wordt na het laden het programma onmiddellijk 'gerund'.
Geeft men een <verschuiving> op (geheel getal) dan wordt het nieuwe startadres gelijk aan het oude + de verschuiving
Categorie: commando
Voorbeeld: `BLOAD "CAS:TEST",R,&H20`

BLOAD "<dev>:[<filenaam>]",S
Als vorige commando maar in dit geval verwijst S naar het video-RAM-geheugen. Let wel <dev> kan hier alleen maar een diskette zijn.
In de SCREEN-modus 5, 6, 7 en 8 wordt de inhoud van de actieve pagina (zie SET PAGE) geladen.
<dev> mag alleen A t/m F zijn.
Categorie: commando
Voorbeeld: `BLOAD "A:TST",S`

BSAVE "<dev>:[<filenaam>]",<beginadres, eindadres>[,<beginadres voor start van het programma>]
Hiermee wordt een in machinecode geschreven programma opgeslagen op een extern medium. De term <dev> en <filenaam> zijn bij BLOAD verklaard. Begin- en eind-adres markeren het geheugengebied dat op het externe medium wordt gekopieerd.

Categorie: commando
Voorbeeld: `BSAVE "CAS:TEST",&HC000,&HE0FF,&HC020`

BSAVE "<dev>:[<filenaam>]",<beginadres>, <eindadres>,S
Als vorige maar nu gaat het om het video-RAM-geheugen. <dev> kan hier alleen maar een diskette zijn.
In de SCREEN-modus 5, 6, 7 en 8 wordt de inhoud van de actieve pagina opgeslagen (zie SET PAGE).
Categorie: commando
Voorbeeld: `BSAVE "A:TST",&H0,&HFFFF,S`

CALL <naam>[(<string>,<string> ...)]
Algemeen commando om bepaalde subroutines die in ROM-cartridge zijn opgeslagen uit te voeren.
De aangegeven strings zijn alfanumerieke constanten, waarmee waarden aan de subroutine kunnen worden doorgegeven. In plaats van de term CALL mag men ook het teken 'underscore' (_) gebruiken.
Categorie: instructie
Voorbeeld: `_TALK`

CALL COM([<X>:],GOSUB<Y>)
Deze instructie kan worden gebruikt als er een RS232C-interface aanwezig is. <X>: geeft het nummer van de interface aan. De 'default'-waarde is 0. <Y> geeft een regelnummer aan. Met deze instructie wordt aangegeven, dat als de RS232C-interface wordt aangesproken, naar de subroutine beginnende met regelnummer <Y> moet worden gesprongen. Deze moet van te voren zijn geactiveerd met de CALL COMON-instructie.
Categorie: instructie
Voorbeeld:

```
10 OPEN "COM:" FOR INPUT AS#1
20 CALL COMON("")
30 CALL COM(,GOSUB 60)
40 PRINT "PROGRAMMA OM GEGEVENS TE ONTVANGEN"
50 GOTO 50
60 PRINT "IK ONTVANG NU:";
70 A$=INPUT$(1,#1)
80 PRINT A$
90 RETURN
100 END
```

CALL COMBREAK ["<n>:"],<specificatiecode>
Hiermee kunnen we zgn. 'break-karakters' overzenden via de RS232C-interface. De opgegeven <specificatiecode> wordt nu gebruikt als teken om de communicatie te verbreken (getal tussen 3-32767). Dit commando kan alleen actief zijn als de RS232C al geopend is.
Met <n> wordt de betreffende RS232C-interface aangeduid. De 'default'-waarde is 0.
Categorie: instructie
Voorbeeld:

```
10 OPEN "COM:" FOR OUTPUT AS#1
20 CALL COMBREAK(,3)
30 CLOSE
40 END
```

CALL COMDTR ["<n:>"],<0 of niet-0>

Hiermee geven we aan dat de RS232C-interface tijdelijk wordt uitgeschakeld, tenminste als de uitdrukking <0 of niet-0> gelijk is aan 'niet-0'. Dit commando geldt alleen als de RS232C al geopend is.
Met <n> wordt de betreffende RS232C-interface aangeduid. De 'default'-waarde is 0.
Categorie: instructie
Voorbeeld:

```
10 OPEN "COM:" FOR OUTPUT AS#1
20 CALL COMDTR(,0)
30 CLOSE
40 END
```

CALL COMINI [([<string exp>][,[<Rx baudsnelheid>][,[<Tx baudsnelheid>][,[<tijdslimiet>]]]])]

Instructie om de RS232C te initialiseren.
Betekenis van de velden:

<string exp>: zie beneden

<Rx baudsnelheid>: baudsnelheid (bits/sec) waarmee de RS232C-interface tekens ontvangt.
Default: 1200 baud

<Tx baudsnelheid>: baudsnelheid (bits/sec) waarmee de RS232C-interface tekens verzendt.
Default: 1200 baud

<tijdslimiet>: tijd (in sec) die de computer zal wachten op de handshake alvorens een fout te melden. Tijdslimiet=0 betekent dat de computer eeuwig blijft wachten.
Default: 0

<string exp>: deze is van de vorm:
"[<knl>:][<bl>[<pa>[<sl>[<x>[<hs>[<ilf>[<slf>[<ss>]]]]]]]]"
Betekenis:

<knl>: kanaalnummer: nummer van de RS232C-interface. Alleen nodig als er meer dan één interface aanwezig is.
Default: 0

<bl>: byte lengte: lengte (in bits) van de in een keer verstuurde gegevenseenheid (byte). In te vullen "5", "7" of "8".
Default: "8"

<pa>: pariteit
- "E" = even pariteit (Even)
- "O" = oneven pariteit (Odd)
- "I" = kijk niet naar pariteit (Ignore)
- "N" = geen pariteit (No)

Default: "E"

<sl>: stop lengte: lengte (in bits) van de pauze tussen het versturen van twee bytes
- "1" = 1 bit
- "2" = 1.5 bits
- "3" = 2 bits

Default: "1"

<x>: XON/XOFF besturing

"X" = wel besturen
"N" = niet besturen

Default: "X"

<hs>: CTS–RTS handshake

"H" = wel handshake
"N" = geen handshake

Default: "H"

```
"[0:][8[n[1[X[H[N[N[N]]]]]]]]"
```

- *Kanaal nr.* Als er maar één kanaal is mag deze aanduiding worden weggelaten. Zgn. "default" is 0
- *Data lengte*
  "5" - 5 bits
  "6" - 6 bits
  "7" - 7 bits
  "8" - 8 bits
- *Pariteitsvlag*
  "E" - Even
  "O" - Odd (oneven)
  "I" - Ignore
  "N" - geen
- *Stop bits (lengte)*
  "1" - 1 bit
  "2" - 1.5 bit
  "3" - 2 bits
- *CTS-RTS handshake*
  "H" - wel handshake
  "N" - geen handshake
- *LF (linefeed) toevoegen* als CR wordt ontvangen
  "A" - toevoegen
  "N" - niet toevoegen
- *LF (linefeed) toevoegen* als CR wordt gezonden
  "A" - toevoegen
  "N" - niet toevoegen
- *Shift in/Shift out control* (niet toegestaan als) datalengte = 7 bits
  "S" - stel control in
  "N" - niet

<ilf>: invoegen van een linefeed-teken elke keer dat er een carriage-return-teken ontvangen wordt

"A" = wel invoegen
"N" = niet invoegen

Default: "N"

<slf>: sturen van een linefeed-teken na elke carriage-return.

"A" = stuur geen linefeed
"N" = stuur wel een linefeed

Default: "N"

<ss>: shift-in/shift-out besturing.

"S" = wel besturen
"N" = niet besturen

Default: "N"

Categorie: instructie
Voorbeeld: `CALL COMINI("0:7N2XHAAN",75,1200)`

CALL COMON("[<n>:]")
CALL COMOFF("[<n>:]")
CALL COMSTOP("[<n>:]")
Met <n> wordt de betreffende RS232C-interface aangeduid.
Door gebruik te maken van een stel CALL-commando's kunt u een BASIC-programma automatisch een subroutine laten uitvoeren zodra via de RS232C-interface een teken ontvangen wordt. De werking van deze commando's is volstrekt analoog aan de werking van ON <gebeurtenis> GOSUB <regelnummer>.
Categorie: instructie
Voorbeeld: zie CALL COM

CALL COMSTAT (["<n>"],<naam variabele>)
Met <n> wordt de betreffende RS232C-interface aangeduid.
Als er een Device I/O-error optreedt bij communicatie via een RS232C-interface wordt aan de variabele een waarde opgegeven waaruit blijkt wat er eventueel is fout gegaan.
De volgende tabel illustreert de mogelijkheden. In een aantal gevallen zijn de standaard Engelse termen overgenomen.

Bit n°	*Betekenis*	
15	Buffer overflow error	0 - geen buffer overflow 1 - wel buffer overflow
14	Time out error	0 - niet 1 - wel
13	Framing error	0 - niet 1 - wel
12	Over run error	0 - niet 1 - wel
11	Parity error	0 - niet 1 - wel

Bit nº	*Betekenis*	
10	Control break-toets is ingedrukt	0 - niet 1 - wel
9	niet gebruikt	
8	niet gebruikt	
7	Clear to Send	0 - niet 1 - wel
6	Timer/counter output-2 0 - timer/counter wordt uitgezet 1 - timer/counter wordt toegekend	
5	niet gebruikt	
4	niet gebruikt	
3	Data Set Ready	0 - niet 1 - wel
2	break detect	0 - niet 1 - wel
1	Ring Indicator	0 - niet 1 - wel
0	Carrier Detect	0 - niet 1 - wel

Categorie: instructie

Voorbeeld:

```
10 OPEN "COM:" FOR INPUT AS#1
20 CALL COMSTAT(,A%)
30 A$="0000000000000000"+BIN$(A%)
40 PRINT RIGHT$(A$,16)
50 CLOSE
60 END
```

CALL COMTERM ["<n>"]

Start de zogenaamde 'terminal emulator mode' van de RS232C-interface. Met <n> wordt de betreffende RS232C-interface aangeduid; de 'default'-waarde is 0. Het RS232C-kanaal moet eerst met CLOSE worden gesloten alvorens CALL COMTERM kan worden uitgevoerd.

De toetsen F6, F7 en F8 hebben een bijzondere betekenis:

F6 schakelt de zgn. 'literal mode' aan/uit. In deze toestand worden de karaktercodes 0 t/m 31 met het teken ∧ afgedrukt, gevolgd door de letter die men krijgt door bij de code 64 op te tellen.

F7 half/full duplex mode. In de 'half duplex mode' worden ingetoetste karakters zowel op het scherm getoond als verzonden via het RS232C-kanaal.

F8 schakelt de zgn. printer-echo aan/uit.

Categorie: commando
Voorbeeld: `CALL COMTERM`

CALL FORMAT
Voor het formatteren van een nog niet gebruikte diskette.
Pas op: een reeds gebruikte diskette zal door het FORMAT-commando volledig gewist worden!
Categorie: commando
Voorbeeld: `CALL FORMAT`

CALL MEMINI[(<X>)]
Reserveert een deel van het geheugen voor het gebruik als memory disk. Geven we een waarde op, dan volgt de hoeveelheid gereserveerde geheugenruimte uit de volgende formule:

INT((<X>-1023)/256+1)*256

De CALL MEMINI moet in ieder geval eenmaal worden gebruikt alvorens we een deel van het geheugen als memory disk kunnen gebruiken. We kunnen de functie van de memory disk weer uitschakelen door het commando CALL MEMINI(0) te geven. De 'device name' (naam van betreffende apparaat) is MEM.
Categorie: instructie
Voorbeeld:
```
CALL MEMINI
SAVE "MEM:PROG.BAS"
```

CALL MFILES
Geeft een overzicht van alle files die in de memory disk zijn opgeslagen. Als van te voren de memory disk niet is geïnitialiseerd door middel van het CALL MEMINI-commando zal CALL MFILES de foutboodschap 'disk offline' veroorzaken.
Categorie: commando
Voorbeeld: `CALL MFILES`

CALL MKILL ("<filenaam>")
Hiermee kunnen we een in de memory disk opgeslagen file wissen.
Categorie: instructie
Voorbeeld: `CALL MKILL("TEST")`

CALL MNAME ("<filenaam1>" AS "<filenaam2>")
Hiermee kunnen we de naam van een in de memory disk opgeslagen file veranderen.
Voorbeeld: CALL MNAME("PROG.BAS" AS "PR1.BAS").
Hierbij wordt aan de file PROG.BAS de nieuwe naam PR1.BAS opgegeven. Als dit commando wordt gegeven, terwijl de memory disk niet is geïnitialiseerd met CALL MEMINI, zal de foutmelding 'disk offline' verschijnen.
Categorie: instructie
Voorbeeld: `CALL NAME("PROG.BAS" AS "PR1.BAS")`

CALL SYSTEM
Commando om MSX-BASIC te verlaten en het systeem onder controle van het MSX-DOS Operating System te krijgen. U moet MSX-DOS dan wel hebben opgestart.
Categorie: commando
Voorbeeld: `CALL SYSTEM`

CDBL(<X>)
Zet <X> over naar een getal met dubbele precisie.
Categorie: functie
Voorbeeld: `PRINT CDBL(7/6)`

CHR$(<ASCII-code>)
Geeft het schriftteken dat behoort bij de <ASCII-code>.
Categorie: functie
Voorbeeld: `PRINT CHR$(65)`

CINT(<X>)
Rondt de waarde van <X> af op een geheel getal (een integer). De waarde moet liggen tussen –32768 en 32767.
Categorie: functie
Voorbeeld: `PRINT CINT(7/6)`

CIRCLE [STEP] (<x,y>),<r>[,[<kleur>]][,[<beginhoek>] [,[<eindhoek>]][,<afplatting>]]]]
Genereert een ellips. Met <x,y> wordt het middelpunt opgegeven, <r> geeft de straal aan. Eventueel kan men slechts een gedeelte tekenen door een begin- en eindhoek op te geven (opgeven in radialen). Laat men STEP weg dan wordt het gewone coördinatenstelsel genomen. Mét STEP schuift het coördinatenstelsel op naar het na STEP genoemde punt.
<afplatting> is een getal dat de verhouding tussen de horizontale en verticale straal weergeeft.
Categorie: instructie
Voorbeeld:

```
10 SCREEN 2
20 CIRCLE (127,95),50,,,1.4
30 GOTO 30
```

CLEAR [<geheugenruimte voor strings>[,<hoogste adres>]]
Hiermee worden alle variabelen gelijk aan 0 gemaakt en bovendien wordt een geheugengebied voor het opslaan van characters (strings, letters) gereserveerd.
Het 'hoogste adres' bepaalt het hoogste adres voor gebruik in BASIC. Ten slotte heeft CLEAR tot gevolg dat alle eventueel nog geopende bestanden gesloten worden. Bedenk dat zonder CLEAR de computer slechts 200 posities reserveert voor het opslaan van letters enz.
We merken bovendien op dat alle voorgeprogrammeerde functies die met behulp van DEF FN zijn aangegeven, alsmede de gestandaardiseerde variabelen die met DEF INT enz. zijn omschreven, tevens worden gewist.
Ook alle tabellen die aan de hand van een DIM-instructie zijn vastgelegd, zullen verloren gaan.
Categorie: instructie
Voorbeeld:

```
10 A=10:B$="TEST"
20 PRINT A,B$
30 CLEAR
40 PRINT A,B$:END
```

CLOAD ["<filenaam>"]
Commando om een programma, eventueel onder naam (max. 6 letters en/of cijfers), van cassette in te lezen. Zo heeft CLOAD "PROG4" tot gevolg dat uw programma onder de

naam PROG4 van cassette wordt ingelezen. Als er iets mis gaat, dient men het laden met CTRL/STOP te onderbreken, de band terug te spoelen en het laden opnieuw te starten.
Laat men de filenaam weg, dan wordt het eerste programma dat op cassette wordt aangetroffen, ingelezen.
Categorie: commando
Voorbeeld: `CLOAD"TEST"`

CLOAD? ["<filenaam>"]
Commando om een programma op cassette te vergelijken met het in het geheugen aanwezige programma. Is alles goed, dan verschijnt "Ok". Als er een fout optreedt, verschijnt "Verify error".
Categorie: commando
Voorbeeld: `CLOAD?"TEST"`

CLOSE [#][<filenummer>][,[#]<filenummer> ...]
Sluit de files met de opgegeven nummers. Geeft men geen nummer op dan worden alle files gesloten.
Categorie: instructie
Voorbeeld:

```
10 MAXFILES=1
20 OPEN "CAS:TEST" FOR OUTPUT AS#1
30 A$="MSX"
40 PRINT#1,A$
50 CLOSE#1
60 END
```

CLS
Veegt het beeldscherm schoon.
Categorie: instructie
Voorbeeld:

```
10 CLS
20 END
```

COLOR[<voorgrond>][,<achtergrond>][,<randgebied>]
Hiermee kan men de kleur van de aangegeven gebieden bepalen. De kleur die verschijnt is de standaardkleur dan wel de kleur die is aangegeven door de COLOR=-instructie (uitzondering: SCREEN-modus 8). De volgende tabel toont de standaardkleuren en tevens hoe deze verkregen zijn door rood, groen en blauw te mengen.

Kleur palet-nummer	*Naam*	*Intensiteit*			*Kleur palet-nummer*	*Naam*	*Intensiteit*		
		rood	*groen*	*blauw*			*rood*	*groen*	*blauw*
0	transparant	0	0	0	8	rood	7	1	1
1	zwart	0	0	0	9	lichtrood	7	3	3
2	groen	1	6	1	10	donkergeel	6	6	1
3	lichtgroen	3	7	3	11	lichtgeel	6	6	4
4	donkerblauw	1	1	7	12	donkergroen	1	4	1
5	lichtblauw	2	3	7	13	magenta	6	2	5
6	donkerrood	5	1	1	14	grijs	5	5	5
7	hemelsblauw	2	6	7	15	wit	7	7	7

SCREEN 6

In deze modus kan men gestreepte patroontjes als randgebied krijgen. Als <randgebied> tussen 0 en 3 ligt, is de kleur overeenkomstig het paletnummer. Voor de waarden 16 tot 31 krijgt men een streepjespatroon volgens de volgende menging:

<randgebied>	*Menging van paletnummers*	*<randgebied>*	*Menging van paletnummers*
16	0 en 0	24	2 en 0
17	0 en 1	25	2 en 1
18	0 en 2	26	2 en 2
19	0 en 3	27	2 en 3
20	1 en 0	28	3 en 0
21	1 en 1	29	3 en 1
22	1 en 2	30	3 en 2
23	1 en 3	31	3 en 3

SCREEN 8

Hier kan het kleurnummer liggen tussen 0 en 255. De kleur wordt bepaald door de formule: kleurnummer=4×R+32×G+B. Hierbij stellen R, G en B de intensiteiten van rood, groen en blauw voor. Voor rood en groen geldt dat de intensiteit moet liggen tussen 0 en 7. Blauw moet liggen tussen 0 en 3. Zo zal een kleur met de intensiteit van rood, groen en blauw van resp. 7,5 en 2, het kleurnummer 190 (4×7+32×5+2) opleveren.
Categorie: instructie
Voorbeeld: `COLOR 5,5,7`

COLOR=(<X>,<XX>,<YY>,<ZZ>)

Hiermee kan men het kleurpaletnummer X een afwijkende intensiteitsverdeling geven (zie COLOR) t.o.v. de standaardmenging. De waarden XX, YY en ZZ vertegenwoordigen de intensiteit (tussen 0 en 7) van respectievelijk rood, groen en blauw. Paletnummer 0 wordt gewoonlijk aan 'transparant' toegewezen: deze kleur is dus niet zichtbaar. Paletnummer 0 kan als de andere paletnummers worden gebruikt door middel van de instructie VDP(9)=VDP(9) OR &H20. Om dit nummer weer als transparant te gebruiken, hanteert men de instructie VDP(9)=VDP(9) AND &HDF.
Categorie: instructie
Voorbeeld: `COLOR=(9,3,7,2)`

COLOR[=NEW]

Met behulp van deze instructie kent men aan de kleur paletnummers weer de standaard intensiteitsverdeling toe (zie COLOR).
Categorie: instructie
Voorbeeld: `COLOR=NEW`

COLOR=RESTORE

Hiermee krijgen de paletnummers de intensiteitsverdeling, die in het videogeheugen is opgeslagen. Deze instructie kan worden gebruikt als de waarden van de paletnummers zijn ingelezen met behulp van de instructie BLOAD...,S. In het videogeheugen is de tabel met de intensiteitsverdeling als volgt opgeslagen:

SCREEN-modus	*videogeheugenlocatie*
SCREEN 0 (40 char.)	&H0400 – &H0420
SCREEN 0 (80 char.)	&H0F00 – &H0F20
SCREEN 1	&H2020 – &H2040
SCREEN 2	&H2020 – &H2040
SCREEN 3	&H2020 – &H2040
SCREEN 4	&H2020 – &H2040
SCREEN 5	&H7680 – &H76A0
SCREEN 6	&H7680 – &H76A0
SCREEN 7	&HFA80 – &HFAA0
SCREEN 8	&HFA80 – &HFAA0

In de SCREEN-modus 5, 6, 7 en 8 kan meer dan een pagina worden ingesteld (zie SET PAGE). De adressen die nu de tabellen van de kleurpaletnummers bevatten, moet men volgens onderstaande berekening bepalen.

SCREEN-modus	*berekening*
SCREEN 5	paginanummer × &H08000 + &H7680
SCREEN 6	paginanummer × &H08000 + &H7680
SCREEN 7	paginanummer × &H10000 + &HFA80
SCREEN 8	paginanummer × &H10000 + &HFA80

Categorie: instructie

Voorbeeld:

```
10 SCREEN 0:WIDTH 40:COLOR 15
20 COLOR=(15,0,7,0)
30 PRINT "XYZ"
40 BSAVE "COL1.PIC",&H400,&H420,S
50 COLOR=NEW
60 FOR I=0 TO 2000:NEXT
70 BLOAD "COL1.PIC",S
80 COLOR=RESTORE
90 END
```

COLOR SPRITE (<X>)=<Y>

Met behulp van deze instructie bepaalt men de kleur van de aangegeven sprite (alleen in SCREEN-modus 4, 5, 6, 7 en 8). X is het spritenummer zoals is vastgelegd met SPRITE$(X). Als Y tussen 0 en 15 ligt, gaat het om een paletnummer. Als we bij dit nummer 32 optellen, zal bij de instructie ON SPRITE GOSUB geen sprong meer worden uitgevoerd als een botsing met een andere sprite plaatsvindt. Tellen we er 64 bij op, dan wordt de volgende functie aangeroepen: negeer de prioriteit en het optreden van een botsing tussen sprites en hanteer een logische OR-operatie om de kleur te bepalen. Hier zal er, als er een botsing met een andere sprite plaatsvindt, geen sprong naar een subroutine volgen.

Categorie: instructie

Voorbeeld:

```
10 SCREEN 5,0
20 B$=""
30 FOR I=1 TO 8:READ A:B$=B$+CHR$(A):NEXT
40 SPRITE$(0)=B$
50 COLOR SPRITE(0)=12
60 FOR I=0 TO 212:PUT SPRITE 0,(I,I),,0:NEXT
70 DATA 24,60,126,255,36,36,66,129
80 END
```

COLOR SPRITE$(<X>)=<X$>
Hiermee kunt u aan een deel van een sprite een kleur toekennen (alleen in SCREEN-modus 4, 5, 6, 7 en 8). X geeft de kleur aan en X$ is een string bestaande uit 1 tot 16 karakters.
Elk karakter correspondeert met een regel (horizontale reep) van de sprite. Als de karaktercode tussen 0 en 15 ligt, gaat het om een paletnummer. Telt u bij de code 32 op, dan ontstaat een situatie waarbij de computer niet op botsingen reageert als dit deel van de sprite met een andere sprite samenvalt (zie voorgaande instructie). Telt u er 64 bij op, dan reageert de computer niet op botsingen en wordt de kleur bepaald door een logische OR (zie ook voorgaande instructie). Telt u er 128 bij op, dan wordt een 'sprite-lijn' 32 pixels naar links verschoven.
Categorie: instructie
Voorbeeld:
```
10 SCREEN 5,0
20 B$=""
30 FOR I=1 TO 8:READ A:B$=B$+CHR$(A):NEXT
40 SPRITE$(0)=B$
50 PUT SPRITE 0,(100,100),,0
60 FOR I=0 TO 2000:NEXT
70 COLOR SPRITE$(0)=CHR$(12)+CHR$(1)+CHR$(130)
80 FOR I=0 TO 2000:NEXT
90 DATA 24,60,126,255,36,36,66,129
100 END
```

CONT
Met dit commando wordt de uitvoering van het programma hervat en wel vanaf de plaats waar het werd onderbroken met CTRL/STOP dan wel met de STOP- of END-instructie
Categorie: commando
Voorbeeld: `CONT`

COPY (<X1>,<Y1>)-(<X2>,<Y2>)[,<XX>] TO(<X3>,<Y3>)[,<YY>[,<bewerking>]]
COPY (<X1>,<Y1>)-(<X2>,<Y2>)[,<XX>] TO <array>
COPY (<X1>,<Y1>)-(<X2>,<Y2>)[,<XX>] TO "[<dev>:]<filenaam>"
COPY <array>[,<richting>] TO(<X3>,<Y3>)[,<YY>[,<bewerking>]]
COPY "[<dev>:]<filenaam>"[,<richting>] TO(<X3>,<Y3>)[,<YY>[,<bewerking>]]
COPY <array> TO "[<dev>:]<filenaam>"
COPY "[<dev>:]<filenaam>" TO <array>
COPY "[<dev>:]<filenaam>" TO "[<dev>:]<filenaam>"

Hiermee kan men een gedeelte van het grafische scherm naar een array of een file kopiëren (Dit werkt alleen in de SCREEN-modus 5, 6, 7 en 8). X1 bepaalt de X-coördinaat van het beginpunt van het deel van het scherm dat men wil kopiëren. Y1 geeft de Y-coördinaat van dat startpunt aan. X2 en Y2 geven de coördinaten van het eindpunt van het te kopiëren gedeelte aan.
X3 en Y3 geven het startpunt aan van het gedeelte waarop de kopie moet worden afgebeeld. In de SCREEN-modus 5, 6, 7 en 8 wordt meer dan een pagina gebruikt (zie SET PAGE). XX geeft dan de 'bron-pagina' aan en YY de 'bestemmingspagina'. Als XX en YY niet zijn aangegeven, wordt uitgegaan van de actieve pagina, d.w.z. de pagina waarin de resultaten van schermbewerkingen zichtbaar zijn.
<richting> geeft de richting aan:

<richting> *verplaatsing*

0	van linksboven naar rechtsonder
1	van rechtsboven naar linksonder
2	van linksonder naar rechtsboven
3	van rechtsonder naar linksboven

<array> is de naam van een array. De grootte van de array volgt uit de volgende formule:

```
INT((<PIXEL>*(ABS(X2-X1)+1)*(ABS(Y2-Y1)+1)+7)/8)+4
```

Hierbij gelden de volgende afspraken:

<pixel> = 4 in de SCREEN-modus 5, 7 en 8 en<pixel> = 2 in SCREEN-modus 6.

<bewerking> is een logische operator. Een uitgebreide bespreking hiervan treft u bij de instructie PSET aan.

<dev> kan alleen slaan op een diskdrive (A t/m F).

Categorie: instructie

Voorbeeld:
```
10 SCREEN 6
20 X1=50:Y1=50:X2=100:Y2=0
30 CIRCLE (X1,Y1),40,3
40 PAINT (X1,Y1),3
50 AA=INT((4*(ABS(X2-X1)+1)*(ABS(Y2-Y1)+1)+7)/8)+4
60 DIM A(AA)
70 COPY (X1,Y1)-(X2,Y2) TO A
80 FOR I=1 TO 2000:NEXT
90 COPY A,2 TO (150,150)
100 FOR I=1 TO 2000:NEXT
110 END
```

COPY SCREEN[<X>]

Instructie om aan te geven dat een aangeboden videosignaal gedigitaliseerd moet worden. X geeft de manier aan waarop dat moet plaatsvinden.

0 digitaliseert het signaal en plaatst dit in de display-pagina (pagina die wordt afgebeeld).

1 digitaliseert 2 frames en plaatst het eerste in de display-pagina en het tweede in de pagina, waarvan het nummer een lager is dan de display-pagina (zie SET PAGE-instructie).

De standaardwaarde is 0.

Categorie: instructie

Voorbeeld: `COPY SCREEN1`

Waarschuwing: deze instructie werkt alleen als de computer voorzien is van een 'video digitizer'.

COS(<X>)

Bepaalt de cosinus van X.

Categorie: functie

Voorbeeld: `PRINT COS(3.1415/6)`

CSAVE "<filenaam>"[,<snelheid in baud>]

Commando om een programma op cassette op te slaan (zie ook CLOAD). Voor de snelheid in baud geldt:

1 1200 baud

2 2400 baud

Laat men deze aanduiding weg dan wordt een snelheid van 1200 baud aangenomen. De baud-snelheid kan ook met SCREEN worden ingesteld.
Categorie: commando
Voorbeeld: `CSAVE "TEST"`

CSGN(<X>)
Zet de waarde van <X> om naar een getal met enkele precisie.
Categorie: functie
Voorbeeld: `PRINT CSNG(9/7)`

CSRLIN
Geeft de y-coördinaat van de positie waarop de cursor zich bevindt.
Categorie: functie
Voorbeeld:

```
10 SCREEN 0:LOCATE 10,20:PRINT CSRLIN
20 END
```

CVI (<2-byte string>)
CVS (<4-byte string>)
CVD (<8-byte string>)
Deze functies decoderen de meegegeven strings naar resp. integer (CVI), enkele precisie (CVS) of dubbele precisie (CVD) waarden. De strings moeten gecodeerd zijn volgens de omgekeerde functies (MKI$, MKS$ en MKD$, zie aldaar) of ingelezen van een random file. We kunnen deze functies gebruiken om getallen terug te vinden die in een random file zijn opgeslagen. In een FIELD-commando kunnen alleen stringvariabelen worden opgegeven; met deze functies kunnen we getallen die in stringvorm gecodeerd zijn uit een record lezen.
Categorie: functie
Voorbeeld: `PRINT CVD(D$);CVS(S$);CVI(I$)`

DATA <n1>[,<n2> ...]
Met deze instructie kan men een lijst met vaste waarden opnemen in een programma die dan één voor één kunnen worden opgehaald met de READ-instructie. De lijst mag zowel uit numerieke waarden bestaan als uit strings (tussen aanhalingstekens (")) en ze moeten door komma's worden gescheiden. READ haalt de waarden sequentieel op. Men kan weer vooraan beginnen met het uitlezen van een DATA-lijst door eerst de RESTORE-instructie uit te voeren.
Categorie: instructie
Voorbeeld:

```
10 READ A,B,C:PRINT A,B,C
20 DATA 5,6,7
30 END
```

DEF FN <functienaam>[(<argumentenlijst>)]=<functie definitie>
Hiermee definieert men eigen functies. De naam waaronder de functie moet worden aangeroepen is de <functienaam> voorafgegaan door FN. Bij string-functies eindigt de naam op $. Eventueel kan men aan de functie argumenten meegeven, die dan worden gebruikt in de <functiedefinitie>. De namen van de argumenten dienen alleen om de functie te definiëren en hebben geen invloed op eventuele variabelen in het hoofdprogramma met dezelfde naam. De argumenten moeten worden gescheiden door een komma.
In de <functiedefinitie> mogen ook variabelennamen voorkomen die niet als argument worden meegegeven in de <argumentenlijst>, maar voor de aanroep van de functie reeds

een waarde hebben gekregen in het hoofdprogramma. De <functiedefinitie> mag maximaal één regel lang zijn.
Categorie: instructie
Voorbeeld:
```
10 DEF FNAB(X,Y)=X^2+Y*Z
20 Z=5
30 I=2:J=3
40 T=FNAB(I,J)
50 PRINT T
60 END
```

DEF <type><letter-aanduiding>
Deze instructie zorgt er voor dat alle variabelen die beginnen met de gegeven <letter(s)> van het opgegeven <type> zijn. De mogelijke <typen> zijn:

INT integer
SGN enkele precisie
DBL dubbele precisie
STR string

Een type-declaratieteken (dus %, #, $) heeft voorrang boven de DEF-instructie. Voorbeelden:

`10 DEFDBL A-E`	Alle variabelen die beginnen met A, B, C, D of E zijn dubbele-precisievariabelen.
`10 DEFSTR A`	Alle variabelen die beginnen met de letter A zijn stringvariabelen.

Categorie: instructie
Voorbeeld:
```
10 DEFINT I
20 I=3/2:PRINT I
30 END
```

DEF USR[<getal>]=<geheugenadres>
Hiermee specificeert men het startadres van een subroutine in machinetaal. <getal> moet tussen 0 en 9 liggen en wordt als naam toegewezen aan de machinetaalroutine. Laat men <getal> weg dan wordt 0 aangenomen. Op grond van dit <getal> en met behulp van de USR-functie kan men nu de desbetreffende subroutine in machinetaal aanroepen. Voorbeeld:
Categorie: instructie
Voorbeeld:
```
10 DEF USR0=1000
20 X=USR0(9*2)
30 END
```

DELETE [<regelnummer>][–<regelnummer>]
Verwijdert alle opgegeven regels. Na uitvoering van dit commando wordt altijd teruggekeerd naar de BASIC-commando-mode.
Categorie: commando
Voorbeeld: `DELETE 10`

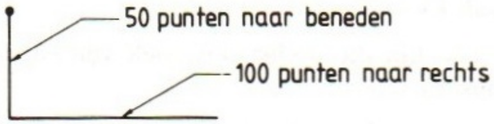
DIM <array-naam>(<maximum index>)[,<array-naam> ...]
Creëert geheugenruimte voor de gespecificeerde array(s) en initialiseert de array-elementen op nul. Wanneer aan een array wordt gerefereerd die niet van tevoren door een DIM-

instructie is gecreeërd, dan wordt 10 als maximum index aangenomen. De laagste index is altijd 0. Met ERASE kunnen we een array wissen.
Categorie: instructie
Voorbeeld:
```
10 DIM A(20)
20 FOR K=1 TO 20:A(K)=K:NEXT K
30 FOR K=1 TO 20:PRINT A(K):NEXT K
40 END
```

DRAW <string>

Met behulp van DRAW kan men allerlei rechte lijnen aangeven met behulp van eenvoudige codes. De codes vormen steeds de string.
Voorbeeld:

```
10 SCREEN 2
20 DRAW "D50R100"
30 GOTO 30
```



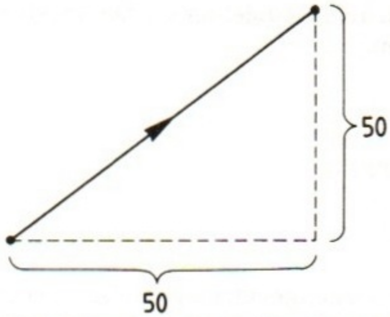
Geeft twee aaneensluitende rechte lijnen. De eerste gaat over 50 beeldpunten naar beneden ("down 50" of afgekort D50) en de tweede 100 punten naar rechts (R100).

Voor ieder lijnstuk mogen we ook een kleur opgeven en wel door middel van de letter C en een kleurcode, bijvoorbeeld:

```
DRAW "C8D50C10R100"
```

De volgende tabel geeft een overzicht van de mogelijkheden:

Code	*Betekenis*
S	Geeft de schaal aan. Na S volgt een getal. De schaal komt dan overeen met getal /4.
A	Na A komt 0, 1, 2 of 3 te staan. Hiermee verdraaien we het coördinaten systeem in stappen van 90°. Zonder A opgave neemt de computer A0 aan.
C	Geeft de kleur aan. Zie bovenstaand voorbeeld.
M	Voor het tekenen van een schuine lijn. Na M komen twee getallen gescheiden door een komma, bijvoorbeeld: M30,50 In dit voorbeeld zal een schuine lijn getrokken worden vanaf de laatste positie, tot het punt dat we krijgen door bij de x-coördinaat 30 en bij de y-coördinaat 50 op te tellen. De waarden x en y mogen ook negatief zijn.
U	U slaat op 'up' dat wil zeggen 'naar boven'. Bijvoorbeeld: U30 geeft aan dat vanaf de laatste positie een lijn naar boven getrokken moet worden en wel over 30 beeldpunten.
D	D slaat op 'down'. Voor het trekken van een lijn naar beneden.
R	R slaat op 'right'. Voor het trekken van een lijn naar rechts.
L	L slaat op 'left'. Voor het trekken van een lijn naar links.
E	Voor een lijn die onder een hoek van 45° naar rechtsboven wordt getrokken. Bijvoorbeeld: E50 geeft als resultaat:



F Voor een lijn die onder een hoek van 45° naar rechtsonder wordt getrokken (zie ook E)

G Voor een lijn die onder een hoek van 45° naar linksboven wordt getrokken (zie ook E)

H Voor een lijn die onder een hoek van 45° naar linksonder wordt getrokken (zie ook E)

Merk op dat lijnen steeds vanaf de laatst bereikte positie worden getrokken. Een vaste beginpositie kunnen we verkrijgen met BMx,y waarin x en y de begincoördinaten voorstellen.
Voorbeeld: `DRAW "BM50,50D30"`

Geeft vanaf punt (50,50) een lijn naar beneden over 30 punten.
Men kan een laatst bereikte punt ook als startpunt voor een nieuwe lijn handhaven en wel door vóór de code een N te plaatsen. Men kan de string ook door middel van een stringvariabele aangeven.
Voorbeeld: ....

```
50 B$="BM50,50D30"
60 DRAW B$
```

Men kan een gedeelte van een string in een DRAW-instructie ook door middel van een variabele weergeven. In het gedeelte van de DRAW-instructie plaatsen we dan een X.
Voorbeeld: ....

```
50 B$="BM50,50D30"
60 DRAW "R100XB$"
```

Ten slotte kan men de getallen die in de DRAW-string voorkomen ook door middel van een variabele aangeven. De variabele komt dan tussen de tekens = en; te staan.
Voorbeeld: ....

```
50 K=30
60 DRAW "B10,10R=K;"
```

Categorie: instructie:
Voorbeeld:

```
10 SCREEN 2
20 DRAW "BM80,80H20"
30 GOTO 30
```

DSKF (<nummer van disk-drive>)
Deze functie geeft de hoeveelheid ruimte die op een diskette over is. De diskdrives zijn als volgt genummerd:

0 = default	2 = B	4 = D	6 = F
1 = A	3 = C	5 = E	

Categorie: functie

Voorbeeld:
```
10 PRINT DSKF(1)
20 END
```

DSKI$ (<nummer van diskdrive>,<nummer van sector>)

Deze functie laadt bij aanroep de aangeduide sector in het geheugendeel dat aangewezen wordt door geheugenplaatsen &HF351 en &HF352.

De nummers 0 t/m 6 geven de diskdrives A t/m F aan.

Categorie: functie

Voorbeeld:
```
10 SCREEN 0:WIDTH 38:COLOR 15,4,4:DEFINT I
20 PRINT "DISKCOPY SECTOR TO SECTOR"
30 PRINT:PRINT
40 PRINT "plaats brondiskette in drive A:"
50 PRINT
60 PRINT "plaats lege diskette in drive B:"
70 PRINT
80 PRINT "druk een toets in"
90 A$=INPUT(1)
100 PRINT:PRINT
110 PRINT "TRACK :"
120 PRINT
130 PRINT "SECTOR:"
140 FOR I=1 TO 719
150 A$=DSKI$(1,I)
160 LOCATE 8,10:PRINT INT(I/9)
170 LOCATE 8,12:PRINT IMOD9
180 DSKO$ 2,I
190 NEXT I
200 END
```

DSKO$<nummer van diskdrive>,<nummer van sector>

Schrijf de inhoud van het geheugen (aangegeven d.m.v. de inhoud van geheugenplaatsen &HF351 en &HF352) naar gespecificeerde sector.

De nummers 0 t/m 6 geven de diskdrives A t/m F aan.

Categorie: instructie

Voorbeeld: zie DSKI$

END

Beëindigt een programma. Deze instructie mag overal in het programma voorkomen. Een END aan het einde van een programma is optioneel, dat wil zeggen niet verplicht. Alle nog geopende files zullen worden gesloten.

Categorie: instructie

Voorbeeld:
```
10 INPUT A
20 IF A<5 THEN END
30 END
```

EOF (<filenummer>)

Bepaalt of het einde van een file is bereikt.

Categorie: functie

Voorbeeld: `IF EOF(2) THEN CLOSE#2`

ERASE <array-naam>[,<array-naam> ...]

Met ERASE kunnen we een array wissen waardoor weer geheugenruimte vrijkomt (zie ook DIM).

Categorie: instructie

Voorbeeld:
```
10 DIM K(100),X$(50)
...
500 ERASE K,X$
```

ERR en ERL

Als in een programma een fout optreedt, geeft ERR het foutnummer van de opgetreden fout (zie appendix foutmeldingen) en bevat ERL het regelnummer waarin de fout is opgetreden.

ERR en ERL kunnen gebruikt worden in door de gebruiker geschreven fout-afhandelingsroutines (zie ook ON ERROR GOTO), meestal in IF...THEN-constructies. Bijvoorbeeld: IF ERR=11 AND ERL=160 THEN ... enz.

Categorie: functie

Voorbeeld:
```
10 ON ERROR GOTO 50
20 A=25:PRINT A
30 B=A/0
40 END
50 PRINT ERL
60 RESUME NEXT
```

ERROR <foutnummer>

Drukt de bij het opgegeven <foutnummer> behorende foutcode af op het scherm. Het <foutnummer> moet groter zijn dan 0 en kleiner dan 255. Voorbeeld: typ in ERROR 11. Op het beeldscherm verschijnt 'division by zero'.

Niet alle foutnummers tussen 0 en 255 worden door BASIC gebruikt. De vrije nummers kan men gebruiken om eigen foutboodschappen te genereren.

Uit het volgende voorbeeld blijkt dat niet altijd de standaard foutboodschap wordt afgedrukt, maar dat we ook een zelf opgestelde boodschap kunnen laten afdrukken.

Categorie: functie

Voorbeeld:
```
10 ON ERROR GOTO 400
20 INPUT "X:-",A
30 IF A>100 THEN ERROR 210
   ...
   ...
400 IF ERR=210 THEN PRINT "MAXIMAAL 100!"
410 RESUME 20
```

EXP(<X>)

Berekent de e-macht van <X>.

Categorie: functie

Voorbeeld: `PRINT EXP(1)`

FIELD [#] <filenummer>,<veldbreedte> AS <stringvariabele>

Hiermee kunnen we toegang krijgen tot de buffer van een random file. We kunnen nu in een BASIC-programma de buffer lezen of beschrijven door gebruik te maken van de stringvariabelen die in het AS gedeelte zijn genoemd.

Categorie: instructie

Voorbeeld: `FIELD #1,20 AS A$,10 AS B$`

Hiermee worden de eerste 20 posities van de buffer van file 1 aan A$ toegewezen en de volgende 10 posities aan B$. Als we nu een GET-commando uitvoeren kunnen we in A$ en B$ het gelezen record opvragen. Een nieuw record kunnen we schrijven door in een LSET of RSET-commando A$ en B$ een nieuwe waarde te geven en vervolgens een PUT-commando uit te voeren. FIELD leest of schrijft zelf niet op de diskette.
De som van de veldbreedtes in een FIELD-commando mag niet groter zijn dan de recordlengte van de file (opgegeven bij het OPEN-commando).

FILES ["<filenaam>"]
Geeft overzicht van de files die zich op diskette bevinden.
Categorie: commando
Voorbeeld: `FILES`

FIX(<X>)
Geeft het gehele deel van <X>.
Categorie: functie
Voorbeeld: `PRINT FIX(1.7)`

FOR ... NEXT
De volledige syntax luidt:

FOR <variabele>=<n> TO <m> [STEP<k>]
instructie 1
instructie 2
...
...
NEXT [<variabele>] [,<variabele> ...]

waarbij n, m en k numerieke uitdrukkingen zijn.
Alle instructies tussen FOR en NEXT worden herhaald uitgevoerd. Net zo vaak totdat de teller (=<variabele>) de waarde van m overschrijdt. De beginwaarde van de teller is n en de teller wordt, iedere keer als alle instructies zijn uitgevoerd, met k verhoogd. Als men geen k specificeert dan wordt de teller steeds met 1 verhoogd.
FOR...NEXT-instructies mogen op hun beurt andere FOR...NEXT-instructies omvatten.
Categorie: instructie
Voorbeeld:

```
10 FOR K=1 TO 10:PRINT K:NEXT K
20 END
```

FRE(Ø) of FRE(<" ">)
Is het argument een Ødan geeft FRE de grootte van de nog niet gebruikte geheugenruimte in aantal bytes. Is het argument een string-variabele, dan geeft FRE het aantal vrije bytes in de geheugenruimte die gereserveerd is voor string-variabelen. Deze laatste ruimte kan met CLEAR worden aangepast.
Categorie: functie
Voorbeeld: `PRINT FRE(0)`

GET [#]<filenummer>[,<recordnummer>]
Hiermee kunnen we een record van een random file inlezen in het buffergeheugen van die file. De inhoud van het record kunnen we vervolgens in een programma gebruiken als er voor de file een FIELD-commando gegeven is, door gebruik te maken van de stringvariabelen die in dat FIELD-commando opgegeven zijn.

Elk record in een random file heeft een nummer. Na het inlezen van een record wordt het nummer van dat record intern opgeslagen, en bij een volgend GET-commando wordt automatisch het volgende record ingelezen. Als er echter in het GET-commando een recordnummer gegeven is, wordt het record met dat nummer gelezen.
Het recordnummer van het laatst ingelezen record kan men vinden met de functie LOC; zie aldaar.
Categorie: instructie
Voorbeeld:
```
10 OPEN "VBD.DAT" AS#1
20 FIELD #1,2 AS A$,10 AS B$
30 FOR K%=1 TO 10
40 GET #1,K%
50 PRINT CVI(A$);B$
60 NEXT
70 CLOSE #1
80 END
```

GET DATE <X$>[,A]
Hiermee kan men de datum die in de klokchip is opgeslagen, opvragen.
Afhankelijk van de MSX-versie is het formaat: MM/DD/YY of DD/MM/YY of YY/MM/DD. (M = month, D = day, Y = year.) Als A is meegegeven, wordt de alarmdatum geretourneerd (de computer kijkt naar DD).
Categorie: instructie
Voorbeeld: `GET DATE PR$:PRINT PR$`

GET TIME<X$>[,A]
Hiermee kan men de in de klokchip bijgehouden tijd opvragen. Het formaat is HH:MM:SS (H = uur, M = minuut, S = seconde). Als A is meegegeven, wordt de alarmtijd geretourneerd.
Categorie: instructie
Voorbeeld: `GET TIME PR$:PRINT PR$`

GOSUB <regelnummer> RETURN <regelnummer>
Deze instructie maakt het mogelijk om naar een subroutine te springen. <regelnummer> is het nummer van de eerste regel van de subroutine. Een RETURN moet ergens in de subroutine voorkomen en zorgt ervoor dat wordt teruggekeerd naar de eerstvolgende instructie na de GOSUB-instructie. Subroutines mogen op hun beurt weer naar andere subroutines verwijzen.
Categorie: instructie
Voorbeeld:
```
10 GOSUB 40
20 GOSUB 40
30 END
40 PRINT "SUBROUTINE"
50 RETURN
```

GOTO <regelnummer>
De programma-uitvoering wordt vervolgd bij het opgegeven <regelnummer>.
Categorie: instructie
Voorbeeld:
```
10 GOTO 30
20 PRINT "A"
30 PRINT "13"
40 END
```

HEX$(<X>)

Geeft een string die de hexadecimale waarde voorstelt van <X>. <X> wordt eerst afgerond naar een geheel getal.
<X> moet liggen tussen – 32768 en 65535.
Categorie: functie
Voorbeeld: `PRINT HEX$(63)`

IF ... THEN

De volledige syntax luidt:

IF <boolean expressie>[<boolean operator><boolean expressie>...]THEN <regelnummer of instructie(s)>[ELSE <regelnummer of instructie(s)>]

N.B. Met 'boolean expressie' wordt bedoeld 'logische voorwaarde' die al of niet juist is. In de <boolean expressies> mogen als relatieteken (operator) voorkomen:

< :kleiner dan
> :groter dan
= :gelijk aan
<= :kleiner dan of gelijk aan
>= :groter dan of gelijk aan
<> :ongelijk aan

Plaatst men na THEN of ELSE meer instructies dan moeten deze gescheiden zijn door een dubbele punt:
De totale uitdrukking moet wel een regel beslaan.
In de uitdrukkingen mogen OR, AND, NOT, XOR, EQV of IMP worden gebruikt (zie Appendix F).
Als de hele expressie (uitdrukking) tussen IF en THEN waar is, worden de instructies na THEN uitgevoerd. Volgt na THEN geen instructie(s) maar een regelnummer dan wordt verder gegaan met de opgegeven regel.
Als de hele expressie tussen IF en THEN niet waar is, worden de instructies na ELSE uitgevoerd of wordt naar de opgegeven regel gesprongen. Is geen ELSE aanwezig dan wordt verder gegaan met de regel die direct volgt op de IF...THEN-instructie.
De instructies na THEN of ELSE mogen weer een IF...THEN-instructie bevatten, mits het geheel op één regel staat. Dus de instructie:

```
IF X>Y THEN PRINT "GROTER" ELSE IF Y>X THEN PRINT "KLEINER" ELSE PRINT "GELIJK"
```

is toegestaan. Iedere ELSE in zo'n instructie wordt gekoppeld aan de dichtstbijzijnde IF waarvoor nog geen bijbehorende ELSE was gevonden.
Categorie: instructie
Voorbeeld:

```
10 INPUT A
20 IF A<3 THEN PRINT "A<3"
30 END
```

INKEY$

Deze functie wordt in de vorm <string-variabele>=INKEY$ gebruikt. De functie controleert of er een toets is ingedrukt op het toetsenbord. Zo ja, dan wordt het betreffende schriftteken aan <string-variabele> toegekend. Zo niet, dan bevat <string-variabele> een lege string (zogenaamde null-string).
Categorie: functie

Voorbeeld:
```
10 PRINT "TYP EEN H: ";
20 A$=INKEY$
30 IF A$<>"H" THEN 20
40 PRINT A$
50 END
```

INP(<X>)
Levert een byte op die wordt ingelezen van inputpoort X.
Categorie: functie
Voorbeeld:
```
10 A=INP(&HA8)
20 A$="00000000"+BIN$(A):PRINT RIGHT$(A$,8)
```

INPUT ["<tekst>";]<variabele 1>[,<variabele 2> ...]
Leest waarden in die door de gebruiker moeten worden ingetoetst op het toetsenbord. Op het scherm verschijnt steeds een vraagteken (?).
Wordt <tekst> meegegeven, dan verschijnt deze tekst vóór het vraagteken op het scherm. De ingetoetste waarden worden toegekend aan de <variabelen>. Er moeten net zoveel waarden, gescheiden door een komma, worden ingetoetst als er variabelen voorkomen in de INPUT-instructie. Ook string-variabelen zijn toegestaan maar er mag geen komma voorkomen in de ingetoetste string. Komt er een fout voor in de ingetoetste waarden dan verschijnt de foutmelding ?Redo en wordt de INPUT-instructie opnieuw uitgevoerd.
Categorie: instructie
Voorbeeld: `10 INPUT A:PRINT A:END`

INPUT #<filenummer>,<variabele 1>[,<variabele 2> ...]
Deze instructie is vergelijkbaar met INPUT maar nu gaat het om de waarden in een file. <filenummer> is het nummer dat aan de desbetreffende file is toegekend met OPEN. De waarden in de file moeten in dezelfde volgorde staan als de variabelen in de INPUT-instructie. Tijdens het inlezen van een string wordt het einde van de string bepaald door een komma, RETURN of line feed. Is het eerste teken van de string echter een aanhalingsteken ("), dan wordt een tweede aanhalingsteken als einde van de string aangenomen.
Categorie: instructie
Voorbeeld:
```
10 OPEN "CAS:DATA" FOR INPUT AS#1
20 IF EOF(1) THEN 40
30 INPUT#1,X$:PRINT X$:GOTO 20
40 CLOSE#1:END
```

INPUT$(<X>)/INPUT$(<X>,[#]<Y>)
Deze functie wordt in de vorm <string-variabele>=INPUT$ (<X>) gebruikt. De functie leest een string in van het toetsenbord van <X> schrifttekens. In plaats van het toetsenbord kan ook file <Y> worden aangegeven. Alle schrifttekens worden geaccepteerd behalve CTRL/C.
Categorie: functie
Voorbeeld: `10 A$=INPUT$(4):PRINT A$:END`

INSTR([I,]<X$>,<Y$>)
Controleert of <Y$> voorkomt in <X$> en geeft dan de positie in <X$>. Komt <Y$> niet voor in <X$> dan geeft INSTR de waarde 0. INSTR zoekt vanaf het begin van <X$> tenzij men met [I,] een andere startpositie opgeeft. Is <Y$> een lege string dan geeft INSTR de waarde 1.
Categorie: functie
Voorbeeld: `10 A$="ABSDEFG":PRINT INSTR(A$,"BCD"):END`

INT(<X>)
Deze functie geeft het grootste gehele getal dat kleiner of gelijk is aan <X>.
Categorie: functie
Voorbeeld: `PRINT INT(3.7)`

INTERVAL ON
INTERVAL OFF
INTERVAL STOP
Hiermee kan men aangeven of een onderbreking die door de ingebouwde klok wordt aangegeven; wordt ingeschakeld, uitgeschakeld of aangehouden. Het interval dient in een ON INTERVAL GOSUB-instructie te worden aangegeven.
Na de instructie INTERVAL ON zal de computer nadat het tijdsinterval is afgelopen naar de subroutine springen die in ON INTERVAL GOSUB is aangegeven.
Categorie: instructie
Voorbeeld:

```
10 ON INTERVAL=300 GOSUB 40
20 INTERVAL ON
30 GOTO 30
40 K=K+6:PRINT K;"SEC"
50 RETURN
```

KEY <n>,"<commandostring>"
Voor het toekennen van een <commandostring> aan een functietoets. <n> is het nummer van de functietoets (1 t/m 10). De <commandostring> moet tussen aanhalingstekens (") staan en mag uit maximaal 15 schrifttekens bestaan. Een RETURN kan men in de <commandostring> opnemen met de toevoeging +CHR$(13).
Categorie: instructie
Voorbeeld: `KEY 1,"RUN"+CHR$(13)`
Wordt nu functietoets 1 gebruikt dan wordt het programma direct uitgevoerd, omdat het commando RUN is afgesloten met een RETURN.

KEY LIST
Geeft een overzicht van de commandostrings van de 10 functietoetsen.
Categorie: instructie
Voorbeeld: `KEY LIST`

KEY ON
KEY OFF
Bepalen of de betekenis van de functietoetsen al of niet op het scherm wordt weergegeven.
Categorie: instructie
Voorbeeld: `KEY OFF`

KEY (<n>) ON
KEY (<n>) OFF
KEY (<n>) STOP
Hiermee kan men nagaan of één van de functietoetsen is ingedrukt. Met ON schakelt men deze toestand in en met OFF schakelt men deze toestand uit.
Met STOP zal de computer niet onmiddellijk naar de subroutine volgens ON KEY GOSUB springen maar dit pas doen als de KEY (n) ON is gegeven.
Categorie: instructie
Voorbeeld:
```
10 KEY(1) ON
20 ON KEY GOSUB 50
30 GOTO 10
40 END
50 PRINT "KEY1":KEY(1)OFF:RETURN
```

KILL "<filenaam>"
Voor het wissen van een file op de diskette.
Categorie: instructie
Voorbeeld: `KILL "A:X1.DAT"`

LEFT$(<X$>,<I>)
Geeft een string die bestaat uit de eerste <I> schrifttekens van <X$>. <I> moet liggen tussen 0 en 255.
Categorie: functie
Voorbeeld: `PRINT LEFT$("MSX",2)`

LEN(<X$>)
Geeft het aantal schrifttekens waaruit <X$> bestaat. Ook spaties worden geteld.
Categorie: functie
Voorbeeld: `PRINT LEN("MSX")`

[LET]<variabele>=<waarde>
Deze instructie kent een waarde aan een variabele toe. De term LET mag eventueel worden weggelaten.
Categorie: instructie
Voorbeeld:
```
10 LET A=14:PRINT A
20 LET A=A+3:PRINT A:END
```

LFILES ["<filenaam>"]
Geeft overzicht van alle files die zich op diskette bevinden. Met LFILES wordt dit overzicht op de printer afgedrukt.
Categorie: commando
Voorbeeld: `LFILES`

LINE[[STEP](<X1>,<Y1>)]-[STEP](<X2>,<Y2>)[,<Z>][,B[F[,<OP>]]]
Tekent een lijn van (X1,Y1) naar (X2,Y2). Met STEP kunnen we aangeven dat het coördinatenstelsel naar het aangegeven punt wordt verplaatst. Geeft men een diagonaal aan en eindigt men deze instructie met B, dan wordt een vierkant met overeenkomstige diagonaal getekend. Met BF wordt het vierkant ingekleurd met het opgegeven kleurnummer Z. Z geeft het paletnummer aan en met <OP> kunnen we eventueel een logische operatie (bijvoorbeeld OR en AND) aangeven, waarmee de kleur wordt bepaald. De invloed van

deze logische operatie komt bij PSET uitvoerig aan de orde. In SCREEN-modus 2, 3, 4, 5 en 8 loopt de X-coördinaat van 0-255. In SCREEN-modus 6 en 7 loopt X van 0-511.
In SCREEN-modus 2, 3 en 4 loopt Y van 0-191 en in SCREEN-modus 5, 6, 7 en 8 van 0-211.
In SCREEN-modus 2, 3, 4, 5 en 7 moet Z tussen tussen 0 en 15 liggen, in SCREEN-modus 6 tussen 0 en 3 en in SCREEN-modus 8 tussen 0 en 255.
Categorie: instructie
Voorbeeld:

```
10 SCREEN 5
20 LINE (0,0)-(511,211)
30 GOTO 30
```

LINE INPUT ["<tekst>";]<stringvariabele>
Leest een string in die wordt ingetoetst op het toetsenbord. De string mag uit alle mogelijke schrifttekens bestaan en een willekeurige lengte hebben tot maximaal 255 tekens.
Categorie: instructie
Voorbeeld: `10 LINE INPUT A$:PRINT A$`

LINE INPUT #<filenummer>,<stringvariabele>
Deze instructie is vergelijkbaar met LINE INPUT, maar leest de string in van een file. <filenummer> is het nummer dat aan de desbetreffende file is toegekend met OPEN. Een string wordt in z'n geheel ingelezen tot aan een RETURN.
Categorie: instructie
Voorbeeld:

```
10 OPEN "CAS:DAT" FOR INPUT AS#1
20 IF EOF(1) THEN 50
30 LINE INPUT#1,A$:PRINT A$
40 GOTO 20
50 CLOSE#1:END
```

LIST [<regelnummer>[–[<regelnummer>]]]
Dit commando geeft de gespecificeerde regels weer op het scherm. Geeft men geen regelnummer op, dan wordt het hele programma weergegeven.
Categorie: commando
Voorbeeld: `LIST`

LLIST [<regelnummer>[–[<regelnummer>]]]
Hetzelfde als LIST, alleen worden de regels nu afgedrukt op de printer.
Categorie: commando
Voorbeeld: `LLIST`

LOAD"<dev>:<programmanaam>"[,R]
Voor het laden van een programma in het geheugen van de computer.
Voor <dev> kan men nemen:
CAS, MEM, A, B, C, D, E, F, COM<nummer>.
Gebruikt men de cassetterecorder dan moet het programma met een SAVE-commando naar cassette zijn weggeschreven. Voegt u ',R' aan het commando toe, dan zal het programma onmiddellijk na het laden worden gestart.
Categorie: commando
Voorbeeld: `LOAD"CAS:DEMO"`

LOC (<filenummer>)
Deze functie geeft een resultaat dat afhankelijk is van de modus waarin de file geopend is. Als het een random file betreft, levert de functie het nummer op van het laatst ingelezen of weggeschreven record: zie GET. Bij andersoortige files levert de functie het aantal behandelde (ingelezen of weggeschreven) bytes op.
Categorie: functie
Voorbeeld:
```
10 OPEN "CAS:DAT" FOR OUTPUT AS#1
20 INPUT A$:PRINT#1,A$:PRINT LOC(1)
30 IF A$<>"END" GOTO 20
40 CLOSE#1:END
```

LOCATE [<X>][,[<Y>][,[<cursor aan/uit>]]]
Verplaatst de cursor naar de aangegeven plaats. De cursor kunnen we aan- of uitzetten door respectievelijk een 1 of een 0 aan te geven.
Geldt alleen voor tekst-modus 1 en 2. Het bereik van X en Y hangt af van een al of niet gegeven WIDTH-instructie.
Categorie: instructie.
Voorbeeld: `10 SCREEN 0:CLS:LOCATE 10,10:PRINT "*"`

LOF (<filenummer>)
Deze functie geeft de lengte (in bytes) van de aangeduide file.
De file die wordt aangeduid moet van te voren wel geopend zijn.
Categorie: functie
Voorbeeld:
```
10 OPEN "A:DAT" FOR INPUT AS#1
20 PRINT LOF(1):CLOSE#1:END
```

LOG(<X>)
Geeft de natuurlijke logaritme van <X>. <X> moet groter zijn dan 0.
Categorie: functie
Voorbeeld: `PRINT LOG(1.5)`

LPOS(<X>)
X kan hier een willekeurig getal zijn. LPOS geeft de positie van de printer in de printbuffer aan
Categorie: functie
Voorbeeld:
```
10 LPRINT:PRINT LPOS(0)
20 LPRINT "MSX";:PRINT LPOS(0)
```

LPRINT [[USING <print formaat>;]<uitdrukking>]
Als PRINT dan wel PRINT USING, maar nu wordt de printer aangestuurd.
Categorie: instructie
Voorbeeld: `LPRINT "MSX"`

LSET <stringvariabele>=<stringexpressie>
RSET <stringvariabele>=<stringexpressie>
Met deze commando's kunnen we gegevens in de buffer van een random file schrijven. LSET zorgt ervoor dat de string aan de rechterkant aangevuld wordt met spaties, zodat de string links in het veld komt te staan. Bij RSET zal de string geheel naar rechts worden verschoven.
Categorie: instructie

Voorbeeld:
```
10 MAXFILES=1
20 OPEN "A:TEST" AS#1
30 FIELD #1,2ASN1$,4ASN2$,8ASN3$,20ASN4$
40 INPUT "A%";A%
50 INPUT "B!";B!
60 INPUT "C#";C#
70 INPUT "D$";D$
80 RSET N1$=MKI$(A%):RSET N2$=MKS$(B!):RSET N3$=MKD$(C#):LSET
   N4$=D$
90 PUT #1,1
100 A%=0:B!=0:C#=0:D$=""
110 PRINT A%;B!;C#;D$
120 GET #1,1
130 A%=CVI(N1$):B!=CVS(N2$):C#=CVD(N3$):D$=N4$
140 PRINT A%;B!;C#;D$
150 CLOSE #1
160 END
```

MAXFILES =<aantal>
Geeft het maximaal aantal files dat tegelijkertijd geopend kan zijn. <aantal> moet tussen 1 en 15 liggen.
Er kunnen 6 files tegelijkertijd op een diskette geopend zijn.
Categorie: instructie
Voorbeeld:
```
10 MAXFILES=2
20 OPEN "CAS:DEMO" FOR INPUT AS#1
30 OPEN "LPT:" FOR OUTPUT AS#2
40 INPUT#1,A$
50 PRINT#2,A$
60 CLOSE
```

MERGE "<dev>:[<filenaam>]"
Voegt een op een extern geheugen opgeslagen programma toe aan de in het geheugen aanwezige programma's. Het programma moet in ASCII-formaat zijn opgeslagen.
De programmaregels worden in oplopende volgorde van het regelnummer geplaatst. Hebben twee regels hetzelfde regelnummer dan wordt de regel in het geheugen vervangen door de regel van het ingelezen programma. Het nieuw ontstane programma blijft in het geheugen staan.
Categorie: commando
Voorbeeld: `MERGE "CAS:PROG1"`

MID$(<X$>,<I>[<,J>])
Geeft een string die een deel is van <X$>. De string begint op positie <I> en is <J> schrifttekens lang. Wordt <J> niet opgegeven dan worden alle schrifttekens vanaf <I> genomen.
Categorie: functie
Voorbeeld: `PRINT MID$("BASIC",2,3)`

MID$(<string 1>,<eerste karakter>[,<aantal karakters>])=<string 2>
Met behulp van dit commando kunnen we <string 1> wijzigen. We geven daartoe aan hoeveel karakters en van welke karakter string 1 veranderd moet worden door i.p.v. het aangegeven gedeelte van string 1, string 2 op te nemen.
Categorie: instructie
Voorbeeld:
```
10 A$= "PHIPLIE"
20 MID$(A$,4,4)="LIPE"
```
Als resultaat zal aan A$ de string "PHILIPE" worden toegekend.

MKI$ (<integer expressie>)
MKS$ (<enkele precisie expressie>)
MKD$ (<dubbele precisie expressie>)
Deze functies leveren een string op die de gecodeerde versie van het meegegeven getal bevat. De string kan gedecodeerd worden met behulp van de omgekeerde functies CVI, CVS en CVD; zie aldaar.
De strings die we krijgen door toepassing van deze functies kunnen in een record van een random file geschreven worden.
Categorie: functie
Voorbeeld:
```
10 RSET D$=MKD$(WD$)
20 RSET S$=MKS$(WS!)
30 RSET I$=MKI$(WI%)
```

MOTOR ON
MOTOR OFF
Hiermee kan men een taperecorder op afstand besturen. MOTOR ON heeft het effect alsof men bij de recorder op de toets "play" drukt en MOTOR OFF schakelt de motor weer uit.
De termen ON en OFF kunnen eventueel worden weggelaten. Er wordt dan van de ene naar de andere toestand geschakeld.
Categorie: instructie
Voorbeeld:
```
10 MOTOR ON
20 CLOAD "DEMO"
```

NAME <oude filenaam> AS <nieuwe filenaam>
Voor het herbenoemen van files op een diskette.
Categorie: instructie
Voorbeeld: `NAME "VBD" AS "EXP1"`

NEW
Heeft tot gevolg dat het in het geheugen opgeslagen programma wordt gewist.
Alle variabelen worden gewist en alle geopende files worden gesloten.
Categorie: commando
Voorbeeld: `NEW`

OCT$(<X>)
Geeft een string die de octale waarde voorstelt van <X>. <X> wordt eerst afgerond tot een geheel getal.
Categorie: functie
Voorbeeld: `PRINT OCT$(636)`

ON ERROR GOTO <regelnummer>
Tengevolge van deze instructie zal, als er een fout optreedt in het programma, de fout niet worden gemeld op het scherm, zoals gebruikelijk, maar zal het programma worden voortgezet op het gespecificeerde <regelnummer>. Daardoor heeft men de mogelijkheid om eigen fout-afhandelingsroutines te schrijven. <regelnummer> is de eerste regel van de fout-afhandelingsroutine. In de routine kan men gebruik maken van de variabelen ERR en ERL.
Met RESUME keert men vanuit de fout-afhandelingsroutine terug naar het hoofdprogramma. De instructie kan weer ongedaan worden gemaakt met ON ERROR GOTO 0. Het verdient aanbeveling om in een fout-afhandelingsroutine de instructie ON ERROR GOTO 0 uit te voeren voor de afhandeling van onvoorziene fouten.
Categorie: instructie
Voorbeeld:
```
10 ON ERROR GOTO 50
20 INPUT "TEKST";A$
30 IF LEN (A$)>5 THEN ERROR 250
40 END
50 IF ERR=250 THEN PRINT "TEKST TE LANG":RESUME 20
60 ON ERROR GOTO 0
70 END
```

ON <integer-expressie> GOTO <regelnummer>[,<regelnummer> ...]
Maakt het mogelijk om de programma-uitvoering voort te zetten op een regelnummer dat wordt bepaald door de waarde van de <integer-expressie>. Levert de <integer-expressie> bijvoorbeeld de waarde 3 op, dan zal het programma worden voortgezet op het derde regelnummer dat na GOTO is gespecificeerd. De waarde van de <integer-expressie> mag niet negatief zijn of groter dan 255.
Als deze waarde 0 is dan wel groter dan het aantal opgegeven regelnummers dan vervolgt de computer het programma met de eerst volgende uitvoerbare instructie.
Categorie: instructie
Voorbeeld:
```
10 INPUT N
20 ON N GOTO 30,40
30 PRINT "1":GOTO 50
40 PRINT "2":GOTO 50
50 END
```

ON <integer-expressie> GOSUB <regelnummer>[,<regelnummer> ...]
Werkt op dezelfde manier als de instructie ON...GOTO, maar nu moeten de opgegeven regelnummers de eerste regel zijn van een subroutine.
Categorie: instructie
Voorbeeld:
```
10 INPUT N
20 ON N GOSUB 40,50
30 GOTO 60
40 PRINT "1":RETURN
50 PRINT "2":RETURN
60 END
```

ON INTERVAL =<tijd> GOSUB <regelnummer>
Hiermee wordt aangegeven dat na een door INTERVAL aangegeven wachttijd, het programma naar de aangegeven subroutine moet springen.
Categorie: instructie
Voorbeeld: zie INTERVAL ON/OFF/STOP

ON KEY GOSUB <regelnummer>[,<regelnummer> ...]
Geeft aan naar welke subroutine gesprongen moet worden als op één van de toetsen F1 t/m F10 wordt gedrukt.
Categorie: instructie
Voorbeeld: zie KEY (<n>) ON/OFF/STOP

ON SPRITE GOSUB <regelnummer>[,<regelnummer> ...]
Geeft aan naar welke subroutine gesprongen moet worden als twee sprites elkaar overlappen.
Categorie: instructie
Voorbeeld: zie SPRITE ON/OFF/STOP

ON STOP GOSUB <regelnummer>[,<regelnummer> ...]
Geeft aan naar welke subroutine gesprongen moet worden als het programma met CTRL/STOP onderbroken wordt.
Categorie: instructie
Voorbeeld: zie STOP ON/OFF/STOP

ON STRIG GOSUB <regelnummer>[,<regelnummer> ...]
Geeft aan naar welke subroutine gesprongen moet worden. Voor regelnummer 1, 2, 3, 4 en 5 wordt gekeken naar:

nr.	*vuurknop of spatiebalk*
1	spatiebalk
2	joystick 1, vuurknop 1
3	joystick 2, vuurknop 1
4	joystick 1, vuurknop 2
5	joystick 2, vuurknop 2

Categorie: instructie:
Voorbeeld: zie STRIG (<X>) ON/OFF/STOP

OPEN "<dev>:[<filenaam>]" [FOR <verwerking>]AS [#]<nummer>
Hiermee wordt een file geopend.
Voor <dev> kan het volgende worden ingevuld:

CAS:	cassetterecorder
CRT:	tekst-beeldscherm
GRP:	grafisch-beeldscherm
LPT:	printer
COM[<Z>]:	RS232-communicatie interface.
A t/m F:	diskdrive 1 t/m 6.
MEM:	memory-disk

Voor <filenaam> mag een naam bestaande uit maximaal 8 karakters plus een extensie van 3 karakters worden genomen (met uitzondering van de cassetterecorder, waar de naam uit maximaal 6 karakters exclusief extensie mag bestaan).
Als men voor <verwerking> OUTPUT neemt, gaat het om een sequentiële file waar gegevens op worden geplaatst. Neemt men INPUT dan gaat het om een sequentiële file van waaruit gegevens worden ingelezen.
Wordt FOR <verwerking> weggelaten dan gaat het om een random file.

Naast OUTPUT en INPUT kan APPEND worden aangegeven. In dat geval gaat het om het uitbreiden van een sequentiële file.
De volgende tabel toont een overzicht van aanduidingen.

<dev>	*OUTPUT*	*INPUT*	*APPEND*	*Weglaten van FOR<verwerking>*
CRT	*			
GRP	*			
LPT	*			
CAS	*	*		
MEM	*	*	*	
A t/m F	*	*	*	*
COM	*	*		*

Met de uitdrukking AS[#]<nummer> wordt een nummer aan het bestand toegekend, dat dan in instructies als INPUT#, PRINT# enz. wordt gebruikt. Ten slotte melden we dat bij de OPEN-instructie verondersteld wordt, dat de aangegeven <dev> ook werkelijk is aangesloten. Anders volgt een foutmelding.
Categorie: instructie
Voorbeelden: zie CLOSE, GET, INPUT# en LINE INPUT#.

OUT <poortnummer, uitdrukking>
Het door <uitdrukking> aangegeven gegeven wordt naar de door <poortnummer> aangegeven poort gestuurd.
Beide aanduidingen moeten liggen tussen 0 en 255.
Categorie: instructie
Voorbeeld: `10 OUT &HA8,INP(&HA8)`

PAD <X>
Met deze belangrijke functie kunnen we nagaan hoe de status is van een aanraakpaneel, lichtpen, muis of 'track ball'.

De volgende tabel toont welk apparaat wordt aangeduid.

X	*Wordt gebruikt bij*
0 t/m 3	Aanraakpaneel aangesloten op joystickconnector 1
4 t/m 7	Aanraakpaneel aangesloten op joystickconnector 2
8 t/m 11	Lichtpen
12 t/m 15	Muis of 'track ball' op joystickconnector 1
16 t/m 19	Muis of 'track ball' op joystickconnector 2

Bij het aanraakpaneel geldt de volgende tabel:

X	*Betekenis van functiewaarde*
0 of 4	bepaalt of paneel is aangeraakt; 0 (niet aanraken) of –1 (aanraken)
1 of 5	X-coördinaat van aangeraakte plaats; PAD (0) of PAD (4) moet –1 zijn
2 of 6	Y-coördinaat van aangeraakte plaats; PAD (0) of PAD (4) moet –1 zijn
3 of 7	bepaalt of schakelaar is ingedrukt (0=niet en –1=wel)

Voor de lichtpen geldt de volgende tabel:

X	*Betekenis van functiewaarde*
8	Bepaalt of pen al of niet gereed is. 0 (niet gereed) of −1 (gereed)
9	X-coördinaat;PAD(8) moet eerst −1 zijn.
10	Y-coördinaat;PAD(8) moet eerst −1 zijn.
11	bepaalt of schakelaar is ingedrukt. (0=Niet en −1=wel)

Voor muis en track ball geldt de volgende tabel:

X	*Betekenis van de functiewaarde*
12 of 16	Geeft altijd −1, maar moet wel voor het bepalen van X of Y worden opgevraagd
13 of 17	X-coördinaat
14 of 18	Y-coördinaat
15 of 19	Geen betekenis

Merk op dat voor alle gevallen, waarbij een X- en Y-coördinaat dient te worden uitgelezen, eerst een specifieke status moet worden gecontroleerd. Zo zal PAD(1) alleen de X-waarde geven als PAD(0) de waarde −1 opleverde. Zodra de specifieke status bereikt is, dient men de coördinaten zo snel mogelijk uit te lezen.
Met behulp van de STRIG-functie kunnen we de status van de muis of track ball bepalen.
Categorie: functie
Voorbeeld:
```
10 SCREEN 2
20 AA=0
30 IF PAD(0)=0 THEN 20
40 X=PAD(1):Y=PAD(2)
50 IF AA=0 THEN PSET(X,Y) ELSE LINE -(X,Y)
60 AA=1
70 GOTO 30
```

PAINT [STEP](<X,Y>)[,[<kleur vlak>][,<kleur rand>]]
Geeft aan dat een bepaald gebied moet worden ingekleurd. Met STEP wordt het coördinatenstelsel verplaatst. Met x en y geven we een punt aan. Het vlak waarbinnen dit punt valt, wordt ingekleurd. Als de randlijn niet volledig ononderbroken is, wordt het gehele scherm ingekleurd.
In SCREEN-modus 2 en 4 moet <kleur vlak> gelijk aan <kleur rand> zijn. In dit geval mag men <kleur rand> niet opgeven. In SCREEN-modus 3, 5, 6, 7 en 8 kan er wel een verschil in deze kleuraanduidingen zijn.
In SCREEN-modus 2, 3, 4, 5 en 7 moeten de kleurpaletnummers tussen 0 en 15 liggen. In SCREEN-modus 6 kan het paletnummer variëren van 0 tot 3 en in SCREEN-modus 8 van 0 tot 255 (zie COLOR).
Categorie: instructie
Voorbeeld:
```
10 SCREEN 7:COLOR 15,4,4
20 CIRCLE (180,180),40,8
30 PAINT (180,180),2,8
40 GOTO 40
```

PDL(<X>)
Geeft de stand van de paddle. <X> mag 1 t/m 12 zijn. De uitkomst van deze functie varieert van 0 tot 255.
Voor <X> = 1, 3, 5, 7, 9 of 11 gaat de computer er vanuit dat de paddle d.m.v. connector 1 is aangesloten. Bij <X> = 2, 4, 6, 8, 10 of 12 wordt aansluiting 2 verondersteld.
Categorie: functie
Voorbeeld: `10 PRINT PDL(1):GOTO 10`

PEEK(<X>)
Geeft de inhoud van geheugenadres X. In de vorm van een decimaal getal moet X tussen –32768 en 65536 liggen. Als X negatief is wordt het zgn. 2-complement gehanteerd: PEEK (–1) = PEEK (65536 – 1).
Categorie: functie
Voorbeeld: `PRINT PEEK(65535)`

PLAY [<eerste kanaal>][,[<tweede kanaal>][,<derde kanaal>]]
Instructie om geluid volgens bepaalde aanwijzingen voort te brengen. PLAY "CDE" laat bijvoorbeeld de tonen C, D en E horen. Vóór de aanduiding van een noot (C, D, E, F, G, A, B en evenzo C+ voor CIS, enz.) kan men letters en getallen plaatsen die een speciale betekenis hebben:

Code	*Betekenis*
On	Geeft betreffende octaaf aan, bijvoorbeeld PLAY "O5CDE": wil zeggen speel C, D en E van octaaf 5. Voor n geldt; $1 \leq n \leq 8$. Zonder nadere aanduiding neemt de computer O4 aan.
Ln	Geeft de duur van een toon aan en wel volgens:

duur	*code*
4 tellen	L1
2 tellen	L2
1 tel	L4
$\frac{1}{2}$ tel	L8
$\frac{1}{4}$ tel	L16
$\frac{1}{8}$ tel	L32
$\frac{1}{16}$ tel	L64

Zonder nadere aanduiding neemt de computer L4 aan.

Nn	Met n tussen 0 en 96: geeft een toon met nummer n aan.
A t/m G	Geeft toon aan. Geeft men bijvoorbeeld A5 aan dan komt dit overeen met O5A, kortom de A bij de vijfde octaaf.
Rn	Geeft rustmaat weer en wel volgens:

duur	*code*
4 tellen	R1
2 tellen	R2
1 tel	R4
$\frac{1}{2}$ tel	R8
$\frac{1}{4}$ tel	R16
$\frac{1}{8}$ tel	R32
$\frac{1}{16}$ tel	R64

Vn — Geeft het volume aan: n=15 komt met 'maximaal volume' overeen. Zonder nadere aanduiding neemt de computer V8 aan.

• — Verlengt de duur van een noot met een factor 1,5.

Sn — Geeft de klankkleur aan (0≤n≤15). Zonder nadere aanduiding neemt de computer S1 aan.

Tn — Geeft het tempo aan. De waarde van n bepaalt het aantal kwartnoten in een minuut. n kan liggen tussen 32 en 255. De standaardwaarde is 120.

Mn — Geeft de mate aan waarin de klankkleur varieert. Zonder nadere aanduiding neemt de computer M255 aan (1≤M≤ 65535).

Categorie: instructie
Voorbeeld: `PLAY "CDECDEEFGC"`

PLAY(<X>)
Geeft informatie over de gebruikte geluidskanalen: 0 = alle kanalen, 1 = kanaal 1, 2 = kanaal 2, 3 = kanaal 3.
Indien het aangegeven geluidskanaal actief is, zal PLAY de waarde –1 geven. In het andere geval krijgt PLAY de waarde 0.
Categorie: functie
Voorbeeld:
```
10 PRINT PLAY(0)
20 PLAY "CDEFG"
30 PRINT PLAY(0):END
```

POINT(<X>,<Y>)
Geeft het kleurnummer van het door de coördinaten X,Y aangegeven punt. Het bereik van X en Y hangt af van de aangegeven grafische modus (zie SCREEN).
Categorie: functie
Voorbeeld: `PRINT POINT(50,50)`

POKE <adres, gegeven>
Plaatst gegeven in aangegeven geheugenadres.
<adres> kan variëren van –32768 tot en met 65535. <gegeven> ligt in het bereik van 0 tot en met 255.
Categorie: instructie
Voorbeeld: `POKE 5000,100`

POS(0)
Geeft de positie van de cursor op de regel. De meest linkse positie is positie 0. 0 is slechts een schijnargument en heeft verder geen betekenis.
Categorie: functie
Voorbeeld: `10 SCREEN 0:LOCATE 10,20:PRINT POS(0)`

PRESET [STEP](<x,y>)[,<kleur>[,<operatie>]]
Zet of wist een stip op (x,y) van het grafische scherm. Als een kleur is opgegeven, is het effect van PRESET gelijk aan dat van PSET. Zonder deze aanduiding wordt een reeds aangegeven punt gewist. Met STEP kan men eventueel het coördinatenstelsel verplaatsen.

Aan de hand van <operatie> kan men de kleur nog beïnvloeden. Bij PSET wordt een volledige opsomming van alle mogelijkheden gegeven.
Categorie: instructie
Voorbeeld:
```
10 SCREEN 2
20 FOR K=1 TO 100:PRESET(K,K),1:NEXT
30 FOR K=1 TO 50:PRESET(K,K):NEXT
40 GOTO 40
```

PRINT [<uitdrukking>[, of;][<uitdrukking><, of;> ...]]
Geeft op het scherm de waarde weer van variabelen, numerieke uitdrukkingen of strings. Strings moeten tussen aanhalingstekens (") staan.
De positie op het scherm waar de gegevens worden geplaatst, wordt bepaald door het scheidingsteken. Is dit scheidingsteken een puntkomma (;) of een spatie, dan worden de gegevens direct achter elkaar geplaatst. BASIC verdeelt een regel in zones van 14 posities breed. Is het scheidingsteken tussen twee gegevens een komma (,) dan wordt het tweede gegeven in de volgende zone geplaatst.
Sluit men de PRINT-instructie af met een komma of puntkomma, dan worden de gegevens bij de volgende PRINT-instructie op dezelfde regel geplaatst, anders op de volgende regel. De term PRINT mogen we ook door een vraagteken aangeven, bijv.:
```
10 ? "UITKOMST";A.
```
Categorie: instructie
Voorbeeld:
```
10 A=2:PRINT"MSX-";A
```

PRINT USING "<notatiecode>";<uitdrukking>[;<uitdrukking> ...]
Deze instructie is een uitbreiding op de PRINT-instructie, waarbij men met behulp van <notatiecode> zelf kan opgeven in welke vorm men de gegevens wil weergeven. De <uitdrukkingen> moeten worden gescheiden door een puntkomma (;).

Bij het weergeven van strings kan men kiezen uit één van de volgende drie <notatiecodes>:

! Alleen het eerste schriftteken van de string wordt weergegeven.

\n spaties Nu worden de eerste 2+n schrifttekens van de string weergegeven.

& De string wordt volledig weergegeven.

Voor het weergeven van numerieke waarden heeft men de keuze uit de volgende speciale <notatiecodes>:

Geeft het aantal posities voor en na de komma aan. Te grote getallen worden afgerond. Te kleine getallen worden door middel van nullen en spaties passend gemaakt. Is het getal te groot voor de gespecificeerde <notatiecode>, dan wordt het procentteken (%) voorafgaand aan het getal weergegeven.
Voorbeelden:
```
PRINT USING "##.##";1.2345
1.23
PRINT USING "##.##";99.996
%100.00
```

\+ Een plusteken vóór of aan het eind van de <notatiecode> zorgt er voor dat een plus- of minteken vóór of aan het eind van het getal wordt weergegeven.

– Een minteken aan het eind van de <notatiecode> zorgt er voor dat bij negatieve getallen een minteken aan het eind wordt weergegeven.

****** Een dubbele asterisk aan het begin van de <notatiecode> zorgt er voor dat eventuele aan het getal voorafgaande spaties met asterisk-tekens worden aangegeven. Bovendien tellen deze twee asterisk-tekens voor twee extra posities.

$$ Een dubbel $-teken aan het begin van de <notatiecode> zorgt er voor dat een $-teken voorafgaand aan het getal wordt weergegeven. De wetenschappelijke notatie (zie verderop) kan niet worden gebruikt in combinatie met $$.

****$** Geeft een combinatie van de twee voorgaande effecten.

, Een komma aan de linkerkant van de decimale punt zorgt er voor dat een komma wordt weergegeven om de drie posities.

Voorbeeld:
```
PRINT USING "####,.##";1234.5
1,234.50
```

Een komma aan het einde van de <notatiecode> wordt gewoon weergegeven.

Voorbeeld:
```
PRINT USING "####.##,";1234.5
1234.50,
```

^^^^ Door vier pijlen aan het eind van de <notatiecode> te plaatsen, wordt het getal in wetenschappelijke notatie weergegeven.

Voorbeeld:
```
PRINT USING "##.##^^^^";123.45
1.23E+02
```

tekst Tekst voor of achter de <notatiecode>, mits van de <notatiecode> gescheiden door een spatie, wordt gewoon weergegeven.

Voorbeeld:
```
PRINT USING "FL ##.## GULDEN";12.34
FL 12.34 GULDEN
```

Categorie: instructie
Voorbeelden: zie hierboven

PRINT # en PRINT # USING

De volledige syntax luidt:

```
PRINT #<filenummer>,[USING"<notatiecode>";]<uitdrukking>[;<uitdrukking>...]
```

Hiermee worden gegevens naar file geschreven. <filenummer> is het nummer dat aan de file is toegekend met OPEN. Voor <notatiecode> gelden dezelfde mogelijkheden als beschreven bij de PRINT USING-instructie. De <uitdrukkingen> worden achter elkaar naar file geschreven en men dient als scheidingsteken alleen de puntkomma (;) te gebruiken. Gebruikt men komma's dan worden ook de spaties, die worden toegevoegd voor het weergeven op het scherm, naar file geschreven.
Ook strings worden achter elkaar geplaatst in de file en zijn daardoor niet meer als afzonderlijke strings herkenbaar. Stel bijvoorbeeld dat A$=JAN en B$=KLAAS, dan schrijft de instructie:

```
PRINT #1,A$;B$
```

als resultaat JANKLAAS naar file. Dit kan alleen maar als één string worden teruggelezen. Men kan dit probleem voorkomen door zelf scheidingstekens tussen te voegen. Bijvoorbeeld:

```
PRINT #1,A$;",";B$
```

heeft als resultaat JAN,KLAAS.
Een andere mogelijkheid is om de strings in twee instructies naar file te schrijven:

```
PRINT #1,A$
PRINT #1,B$
```

Door de PRINT-instructie niet met puntkomma af te sluiten, wordt automatisch een RETURN toegevoegd aan het eind van de string. De RETURN fungeert nu als scheidingsteken.
Wanneer de string zelf een komma bevat, wordt deze bij het teruglezen als twee strings beschouwd. Bijvoorbeeld A$=JAN,KLAAS en de instructie

```
PRINT #1,A$
```

levert in de file JAN,KLAAS op en zal na inlezen met:

```
INPUT #1,A$,B$
```

als resultaat opleveren dat A$=JAN en B$=KLAAS. Dit probleem kan men vermijden door de string tussen aanhalingstekens (") in de file te plaatsen (zie ook de INPUT-instructie). De instructie wordt nu:

```
PRINT #1,CHR$(34);A$;CHR$(34).
```

34 is de ASCII-code voor aanhalingstekens (").
Categorie: instructie
Voorbeeld: zie hierboven

PSET [STEP](<x,y>) [,<Z>[<,operatie>]]
Hiermee plaatst men een stip op het grafische scherm. De coördinaten van die stip komen met X, Y overeen. De waarden die X en Y kunnen aannemen, hangen af van de gekozen grafische modus (zie SCREEN). Met Z wordt het kleurpaletnummer aangegeven. In SCREEN-modus 2, 3, 4, 5 en 7 kan Z variëren van 0 tot 15. In SCREEN-modus 6 kan Z variëren van 0 tot 3 en in SCREEN-modus 8 van 0 tot 255 (zie COLOR). Zonder nadere aanduiding wordt voor Z de waarde 15 aangenomen. In SCREEN-modus 5 t/m 8 kan men de kleur mede laten afhangen van een aangegeven operatie.
De volgende tabel toont hoe het uiteindelijke kleurpaletnummer (C) wordt bepaald door de logische operatie met het opgegeven kleurpaletnummer Z en het kleurpaletnummer (S) van het opgegeven beeldschermpunt:

Aangegeven operatie	*C wordt door de volgende logische bewerking bepaald*
XOR	C=NOT (Z)*S+Z*NOT(S)
OR	C=Z+S
AND	C=Z*S
PSET	C=Z
PRESET	C=NOT (Z)

Daarnaast kan men nog de aanduidingen TXOR, TOR, TAND, TPSET en TPRESET gebruiken. Deze hebben hetzelfde effect als de overeenkomstige termen zonder T, met het verschil dat de transparante kleur geen effect heeft.
We merken hierbij nog op dat het effect van XOR, OR en AND eenvoudig met een PRINT-instructie kan worden bepaald. Bijvoorbeeld voor Z=1, S=2 en AND: PRINT 1 AND 2.
Categorie: instructie
Voorbeeld:

```
10 SCREEN 2
20 FOR K=1 TO 100:PSET (K,K):NEXT
30 GOTO 30
```

PUT [#]<filenummer>[,<recordnummer>]

Hiermee kunnen we het record dat in het buffergeheugen van de aangegeven file staat wegschrijven naar de file. Het nummer waaronder het record weggeschreven wordt is of 1 groter dan het nummer van het laatst gelezen óf geschreven record (als er geen recordnummer is opgegeven), óf het is gelijk aan het opgegeven nummer.
Zie ook FIELD, GET, LSET, RSET en OPEN
Categorie: instructie
Voorbeeld:

```
10 OPEN "EXPL.DAT" AS#1
20 FIELD #1,2 AS A$,10 AS B$
30 FOR K%=1 TO 10
40 INPUT N%,S$
50 LSET A$=MKI$(N%)
60 RSET B$=S$
70 PUT #1,K%
80 NEXT
90 CLOSE #1
100 END
```

PUT SPRITE <vlaknummer>[,[STEP](x,y)][,<kleur>] [,<sprite-nummer>]

Plaatst sprite met aangegeven kleur en nummer op positie (x,y).
<vlaknummer> is het prioriteitsnummer van de sprite, 0 is de hoogste prioriteit; 31 is laagste prioriteit.
Indien 2 sprites op dezelfde positie op het beeldscherm staan, is de sprite met de hoogste prioriteit zichtbaar.
Met STEP kan men eventueel het coördinatenstelsel verplaatsen. Voor een 8×8-sprite kan men als <sprite-nummer> een getal tussen 0 en 255 nemen. Voor een 16×16-sprite een getal tussen 0 en 63.
Aan de hand van de COLOR SPRITE-instructie kunt u de kleur van de sprite nog beïnvloeden. In SCREEN-modus 2 en 3 kunnen slechts 4 sprites naast elkaar (op een regel) worden afgebeeld. In SCREEN-modus 4, 5, 6 en 7 kunt u 8 sprites naast elkaar afbeelden.
Categorie: instructie
Voorbeeld:

```
10 CLS:COLOR,11,11:SCREEN 2
20 A$"":FOR K=1 TO 8:A$=A$+CHR$(16):NEXT
30 SPRITE$(1)=A$:PUT SPRITE 0,(40,40),1,1
40 GOTO 40
```

READ <variabele>[,<variabele> ...]
READ wordt altijd gebruikt in combinatie met DATA en kent de door DATA gespecificeerde waarden toe aan de opgegeven <variabelen>. Zie verder DATA.
Categorie: instructie
Voorbeeld: zie DATA

REM <commentaar>
Hiermee is het mogelijk commentaar aan een programma toe te voegen. Alle informatie na REM wordt door BASIC genegeerd. REM kan worden afgekort met het symbool '.
Categorie: instructie
Voorbeeld:

```
10 REM BEETHOVEN
20 PLAY "GR8GR8GR8L2D+"
```

RENUM [<nieuw regelnummer>][,[<oud regelnummer>][,[<stapgrootte>]]]
Dient voor het opnieuw nummeren van de regels. <nieuw regelnummer> is het beginnummer van de eerste regel. Specificeert men dit niet dan wordt met 10 begonnen. <oud regelnummer> geeft aan vanaf welke regel het hernummeren moet beginnen. Geeft men geen <oud regelnummer> op dan wordt met de eerste regel begonnen. <stapgrootte> is het getal waarmee het regelnummer steeds moet worden verhoogd. Standaard is dat 10.
Categorie: commando
Voorbeeld: `RENUM 50,10,20`

RESTORE [<regelnummer>]
Hiermee kunnen lijsten met waarden die door DATA zijn gespecificeerd weer van het begin af aan met READ worden gelezen. Zie verder DATA. Eventueel kan men met <regelnummer> een betreffende regel met DATA opgeven.
Categorie: instructie
Voorbeeld:

```
10 READ A,B,C:PRINT A,B,C
20 RESTORE
30 READ D,E:PRINT D,E
40 DATA 5,6,7
```

RIGHT$(<X$>,<I>)
Geeft een string die bestaat uit de laatste <I> schrifttekens van <X$>.
Categorie: functie
Voorbeeld: `PRINT RIGHT$("MSX",2)`

RND(<X>)
Geeft een random-getal tussen 0 en 1. Welke serie random-getallen wordt gegenereerd hangt af van het begingetal.

<X> = negatief er wordt vooraan begonnen van de serie random-getallen;
<X> = positief geeft het volgende random-getal in de serie;
<X> = 0 geeft het laatste random-getal nogmaals.
Met RND (– TIME) krijgen we werkelijk iedere keer andere getallen (vergelijk met RANDOMIZE van andere BASIC-versies).

Categorie: functie
Voorbeeld: `10 FOR K=1 TO 100:PRINT RND(1):NEXT`

RSET
zie LSET

RUN[<X>]
RUN"[<dev>:]<programmanaam>"[,R]
Met RUN kunnen we een programma starten. RUN X geeft aan dat een in het geheugen aanwezig programma vanaf regel X dient te worden uitgevoerd. Geven we <dev> op, dan geven we daarmee aan dat het programma zich op een extern medium bevindt. Voor <dev> kunnen we invullen: CAS, MEM, A t/m F en COM [<n>].
RUN heeft tevens tot gevolg dat alle nog geopende files zullen worden gesloten, tenzij R wordt opgegeven.
Categorie: commando
Voorbeeld: `RUN`

SAVE"[<dev>:]<programmanaam>"[,A]
Hiermee zet men een programma uit het geheugen op het aangegeven medium (zie LOAD). Voor <dev> mag men de volgende aanduidingen plaatsen: CAS, MEM, A t/m F en COM [<n>].
De naam van het programma is tevens de naam, die we bij LOAD en MERGE moeten gebruiken om het programma weer op te roepen. Met ,A geven we aan dat de file in ASCII-formaat dient te worden opgeslagen.
Categorie: commando
Voorbeeld: `SAVE "CAS:DATA"`

SCREEN[<X>[,<Y>[,<Z>[,<XX>[,<YY>[,<ZZ>]]]]]]
Met de SCREEN-instructie wordt aangegeven hoe het scherm moet worden gebruikt. Als de computer wordt aangezet, wordt automatisch SCREEN0,0,1,1,0,0 aangenomen.
De letteraanduidingen hebben de volgende betekenis:

<X>	*modus:tekst of grafisch*
0	tekstmodus 1 met WIDTH 40 : 40 kolommen × 24 regels
	met WIDTH 80 : 80 kolommen × 24 regels
1	tekstmodus 2 met WIDTH 32 : 32 kolommen × 24 regels
2	grafische modus 1: 256×192 pixels
3	grafische modus 2: 64× 48 pixelblokken
4	grafische modus 3: 256×192 pixels
5	grafische modus 4: 256×212 pixels
6	grafische modus 5: 512×212 pixels
7	grafische modus 6: 512×212 pixels
8	grafische modus 7: 256×212 pixels

Het aantal verschillende kleuren op een beeldscherm is afhankelijk van de SCREEN-modus (<X>) en wel als volgt:

<X>	*Kleur*	*<X>*	*Kleur*
0	2 uit 512 kleuren	5	16 uit 512 kleuren
1	2 uit 512 kleuren	6	4 uit 512 kleuren
2	16 uit 512 kleuren	7	16 uit 512 kleuren
3	16 uit 512 kleuren	8	256 kleuren
4	16 uit 512 kleuren		

De volgende instructies kunt u alleen in een grafische modus gebruiken: CIRCLE, COLOR SPRITE, COLOR SPRITE$, COPY, DRAW, LINE, PAINT, PSET, PRESET, ON SPRITE GOSUB, SPRITE ON/OFF/STOP, POINT en PUT SPRITE.

<Y>	*Formaat van de sprites*
0	8×8, zonder vergroting
1	8×8, met vergroting (factor 2)
2	16×16, zonder vergroting
3	16×16, met vergroting (factor 2)

<Z>	*Wel of geen geluid bij indrukken toets*
0	Geeft geen piepsignaal bij indrukken toets
1	Geeft piepsignaal bij indrukken toets

<XX>	*Snelheid bij in/uitvoer van cassette*
1	1200 baud
2	2400 baud

<YY>	*Type printer*
0	MSX-printer
1	Anders

<ZZ>	*Display (weergave) modus*
0	Normaal
1	Interlaced (zelfde pagina)
2	Normaal: achtereenvolgens even en oneven pagina
3	Interlaced: achtereenvolgens even en oneven pagina

In display-modus 2 en 3 dient het nummer van de af te beelden pagina oneven te zijn. De even pagina die daarna (afwisselend) wordt getoond, krijgt men door van de oneven pagina 1 af te trekken (zie SET PAGE).

Categorie: instructie
Voorbeeld: zie COLOR SPRITE, LINE, PSET

SET ADJUST(<X>,<Y>)

Met behulp van dit commando kunnen we het beeld iets verplaatsen. Dit kan van belang zijn als onze monitor op TV het beeld niet goed centreert. X en Y mogen variëren tussen −7 en 8. De aangegeven waarden worden in de klokchip opgeslagen en worden zodoende bewaard, zelfs als de computer wordt uitgeschakeld.
Categorie: instructie
Voorbeeld: `SET ADJUST(3,2)`

SET BEEP<X>,<Y>

Hiermee kunnen we het geluid van het pieptoontje dat de computer automatisch bij veel situaties gebruikt, instellen. X mag variëren van 1 tot 4 en bepaalt de toonhoogte. Y bepaalt het volume en mag ook variëren van 1 tot 4.
De aangegeven waarden worden in de klokchip opgeslagen en worden zodoende bewaard, zelfs als de computer wordt uitgeschakeld.
Categorie: instructie
Voorbeeld: `SET BEEP 3,2`

SET DATE<X$>[,A]
Hiermee kunnen we de datum opgeven. Het formaat waarop dat dient te geschieden is:

MM/DD/YY of DD/MM/YY of YY/MM/DD

en wel afhankelijk van de MSX-landenversie. Met A kunnen we aangeven dat er een alarmsignaal moet worden aangegeven (in combinatie met SET TIME...,A). De computer kijkt, indien A is aangegeven, alleen naar de datum.
De aangegeven string wordt in de klokchip opgeslagen en de datum wordt continu bijgehouden, zelfs als de computer uit staat.
Categorie: instructie
Voorbeeld: `SET DATE "12/02/86"`

SET PAGE <X>,<Y>
Om deze instructie te begrijpen dienen we te weten dat het videogeheugen in een aantal stukken is verdeeld. Bij een 128K videogeheugen zijn dat afhankelijk van de grafische modus 2 of 4 stukken (pagina's) die we nummeren met 0, 1, 2 en 3. De afbeelding die we zien is steeds de afbeelding van maar één pagina. Zonder nadere aanduiding is dit altijd pagina 0. Met SET PAGE X, Y geven we aan dat pagina X wordt afgebeeld en dat de instructies waarmee we een nieuwe afbeelding opbouwen in pagina Y worden opgeslagen. Deze instructie is alleen van toepassing in SCREEN-modus 5, 6, 7 en 8. Voor 128K videogeheugen gelden de volgende afspraken:

SCREEN-modus	*Pagina's*
5	0, 1, 2 en 3
6	0, 1, 2 en 3
7	0 en 1
8	0 en 1

Categorie: instructie
Voorbeeld:
```
10 SCREEN 5:CLS
20 LINE (0,0)-(100,100),1,BF
30 SET PAGE 0,1:LINE (100,100)-(200,200),1,BF
40 SET PAGE 1,0:SET PAGE 0,1:GOTO 40
```

SET PASSWORD<X$>
Hiermee kunnen we de computer een password opgeven. Een gegeven dat ook na het uitzetten van de computer wordt onthouden. Als een 'password' is opgegeven, zal de computer bij het opstarten van het systeem altijd om dit 'password' vragen. Zo verkrijgt men dus een beveiliging tegen ongewenst gebruik. X$ stelt het 'password' voor, het gaat om een string van max. 255 karakters.
Van de drie instructies SET PASSWORD, SET PROMPT en SET TITLE onthoudt de computer slechts één gegeven en wel het gegeven dat behoort bij de laatst gegeven instructie.
Categorie: instructie
Voorbeeld: `SET PASSWORD "MSX"`

SET PROMPT <X$>
Na afloop van ieder commando ziet men steeds "Ok". Dit is de zgn. 'prompt'. Met deze instructie kunnen we deze term door de string X$ vervangen. Zie ook SET PASSWORD.
Categorie: instructie
Voorbeeld: `SET PROMPT "READY"`

SET SCREEN
Hiermee kan men een aantal parameters van de SCREEN-informatie continu opslaan. Daarna zal bij het opstarten van het systeem steeds worden uitgegaan van deze gekozen parameters.
De volgende tabel toont de mogelijkheden:

SCREEN-modus	0 of 1
WIDTH (aantal kolommen)	1 t/m 80
Voorgrondkleur-paletnummer	0 t/m 15
Achtergrondkleur-paletnummer	0 t/m 15
Randgebiedkleur-paletnummer	0 t/m 15
Weergave van functietoetsen	ON of OFF
Klikgeluid bij intoetsen	ON of OFF
Printer-modus	MSX of niet-MSX
Cassette-baudrate	1200 of 2400
Display-modus	0 t/m 3

Categorie: instructie
Voorbeeld:
```
10 SCREEN 0:WIDTH 80:KEY OFF
20 SET SCREEN:END
```

SET TIME<X$>[,A]
Hiermee kunt u de tijd van de klokchip instellen en wel volgens het formaat HH:MM:SS. Als ,A is opgegeven, worden HH en MM als alarmtijd opgevat. De tijd wordt automatisch bijgehouden, ook als het systeem uit staat.
Categorie: instructie
Voorbeeld: `SET TIME "11:55:06"`

SET TITLE <X$>[,<Y>]
Hiermee kan men bij het opstarten van het systeem steeds de titel X$ in de kleur <Y> op het scherm weergeven. (Zie SET PASSWORD).
Wanneer <X$> 6 karakters lang is zal het systeem bij het opstarten wachten totdat een toets wordt ingedrukt.
Categorie: instructie
Voorbeeld: `SET TITLE "HALLO"`

SET VIDEO<X>[,<Y>[,<Z>[,<XX>[,<YY>[,<ZZ> [<XXX>]]]]]]
Hiermee kan men de zogenoemde superimpose-modus aangeven. Superimpose wil zeggen, dat er beelden kunnen worden gemengd (gesuperponeerd). De volgende tabellen geven de mogelijkheden aan:

<X> *Waar komt het beeld vandaan?*
0 Van de computer
1 Van de computer
2 Te zamen met een ander beeld (superimposed)
3 Van de video-input

In 0 is er geen externe synchronisatie mogelijk en in 1, 2 of 3 is er geen 'composite output' (gemengde beelden of output) mogelijk.

<Y> *Hoe groot is de intensiteit?*
0 Half
1 Vol

Zonder nadere aanduiding is Y gelijk aan 0.

<Z> *Regelt de kleurenbesturing*
0 Alleen voor output (uitvoer)
1 Alleen voor input (invoer)

Zonder nadere opgave wordt 0 aangenomen.

<XX> *Regelt de synchronisatie*
0 Intern
1 Extern

Zonder nadere opgave wordt 0 aangenomen.

<YY> *Regelt het audiosignaal*
0 Alleen van computer
1 Meng externe input van rechterkanaal met computer
2 Meng externe input van linkerkanaal met computer
3 Meng externe kanalen met computer

Zonder nadere opgave wordt 0 aangenomen.

<ZZ> *Regelt de externe video-input*
0 RGB-euroconnector
1 TV-connector

Zonder nadere aanduiding wordt 0 aangenomen.

<XXX> *Selectie van audio/video-output van RGB-euroconnector*
1 Wordt gekozen
0 Wordt niet gekozen

Zonder nadere aanduiding wordt 0 aangenomen.
Categorie: instructie
Voorbeeld: `SET VIDEO 2`
Waarschuwing: deze instructie werkt alleen als de computer is voorzien van 'super imposed'.

SGN(<X>)
Bij <X> = positief wordt SGN(<X>) = 1
Bij <X> = 0 wordt SGN(<X>) = 0
Bij <X> = negatief wordt SGN(<X>) = –1
Categorie: functie
Voorbeeld: `PRINT SGN(31)`

SIN(<X>)
Geeft de sinus van <X>. <X> moet worden opgegeven in radialen.
Categorie: functie
Voorbeeld: `PRINT SIN(3.1415/12)`

SOUND <registernaam, getal>
Instructie om bepaald geluid te genereren. De volgende afspraken gelden:

register	*bereik getal*	*betekenis*
0	0–255	frequentie kanaal A
1	0–15	frequentie kanaal A
2	0–255	frequentie kanaal B
3	0–15	frequentie kanaal B
4	0–255	frequentie kanaal C
5	0–15	frequentie kanaal C
6	0–31	frequentie ruis
7	0–63	keuze kanaal: toon en ruis
8	0–15	volume kanaal A
9	0–15	volume kanaal B
10	0–15	volume kanaal C
11	0–255	frequentie van patroonvariatie
12	0–255	frequentie van patroonvariatie
13	0–14	keuze patroon

Categorie: instructie
Voorbeeld: `10 FOR K=1 TO 10:SOUND K,0:NEXT K`

SPACE$(<X>)
Geeft een string bestaande uit <X> spaties. <X> wordt op een geheel getal afgerond en moet tussen 0 en 255 liggen.
Categorie: functie
Voorbeeld: `10 A$=SPACE$(20):PRINT A$;"A"`

SPC(<X>)
Geeft <X> spaties weer op het scherm of printer. SPC mag alleen worden gebruikt binnen een PRINT- of LPRINT-instructie. <X> moet liggen tussen 0 en 255.
Categorie: functie
Voorbeeld: `PRINT "MSX";SPC(3);"2"`

SPRITE ON
SPRITE OFF
SPRITE STOP

Met ON, OFF en STOP wordt aangegeven of de computer attent is op het optreden van 'botsende' sprites. Zie ON SPRITE GOSUB. Met SPRITE STOP wordt aangegeven dat de 'onderbrekings-toestand' moet worden aangehouden. In dat geval zal pas naar de door ON SPRITE GOSUB aangegeven subroutine gesprongen worden als weer SPRITE ON wordt aangetroffen.

Categorie: instructie
Voorbeeld:

```
10 DATA 60,66,165,129,165,153,66,60
20 DATA 60,126,219,255,255,219,102,60
30 A$=""
40 FOR I=1 TO 8
50 READ A:A$=A$+CHR$(A)
60 NEXT
70 B$=""
80 FOR I=1 TO 8
90 READ A:B$=B$+CHR$(A)
100 NEXT
110 SCREEN 2,1:COLOR 15,4,1
120 ON SPRITE GOSUB 210
130 SPRITE$(0)=A$:SPRITE$(1)=B$
140 SPRITE ON
150 A=INT(RND(1)*256):B=INT(RND(1)*256)
160 FOR I=0 TO 191
170 PUT SPRITE 0,(A,I),1
180 PUT SPRITE 1,(B,191-I),15
190 NEXT:GOTO 140
200 SPRITE OFF
210 PLAY "L4CDEFEDCREFGAGFER"
220 PUT SPRITE 0,(0,208)
230 PUT SPRITE 1,(0,208)
240 I=191:RETURN
```

SPRITE$

Het gaat hier om een systeemvariabele die steeds in de volgende vorm wordt gebruikt: SPRITE$(<nummer>)=(<string-uitdrukking>). Het <nummer> geeft het z.g. sprite-nummer aan. Afhankelijk van de definitie van het sprite-formaat zoals d.m.v. de SCREEN-instructie gegeven, kan <nummer> maximaal 63 dan wel 255 zijn. Met behulp van <string-uitdrukking> geven we de vorm van de sprite aan. Gewoonlijk doen we dit aan de hand van een aantal CHR$-functies, bijv.

```
SPRITE$(1)=   CHR$(&H18) + CHR$(&H3C) + CHR$(&HFF) + CHR$(&H99) + CHR$(&H99)
              + CHR$(&HFF) + CHR$(&HC3) + CHR$(&HFF)
```

Voor een 8 × 8 sprite heeft men zo 8 CHR$-functies nodig.
SPRITE$ kan worden gebruikt in de verschillende grafische modi (zie ook PUT SPRITE).
Voor het kleuren van sprites zie COLOR SPRITE en COLOR SPRITE$.
Categorie: systeemvariabele
Voorbeeld: zie SPRITE ON/OFF/STOP

SQR(<X>)
Geeft de wortel uit <X>. <X> moet groter dan of gelijk zijn aan nul.
Categorie: functie
Voorbeeld: `PRINT SQR(4)`

STICK(<X>)
Geeft de stand van de joystick (1 of 2). Vult men voor X de waarde 0 in, dan wordt gekeken naar de cursor-toetsen.
Zie voor de richtingen afbeelding hiernaast.



Categorie: functie
Voorbeeld: `PRINT STICK(0)`

STOP
Onderbreekt het uitvoeren van een programma. We kunnen de uitvoering van het programma dan weer met het CONT-commando vervolgen.
STOP kan overal in het programma worden geplaatst.
Categorie: instructie
Voorbeeld:

```
10 INPUT A
20 PRINT A:IF A=0 THEN STOP
30 GOTO 10
```

STOP ON
STOP OFF
STOP STOP
Regelt de modus bij een door CTRL/STOP veroorzaakte onderbreking (zie ON STOP GOSUB). Bij STOP STOP wordt de onderbrekings-toestand aangehouden d.w.z. dat pas naar de door ON STOP GOSUB aangetroffen subroutine wordt gesprongen als weer STOP ON wordt aangetroffen.
Categorie: instructie
Voorbeeld:

```
10 ON STOP GOSUB 50
20 STOP ON
30 INPUT A$
40 IF A$="END" THEN STOP OFF:END ELSE GOTO 30
50 PRINT "TOETS END (+ RETURN)":RETURN
```

STRIG(<X>)
Geeft aan of op vuurknop van joystick of spatiebalk wordt gedrukt (−1=wel en 0=niet)

<X>	*betekenis*
0	spatiebalk
1 of 3	joystick 1
2 of 4	joystick 2

Categorie: functie
Voorbeeld: `PRINT STRIG(0)`

STRIG (<x>) ON
STRIG (<x>) OFF
STRIG (<x>) STOP

Regelt de modus bij een door de joystick of spatiebalk veroorzaakte onderbreking (zie ON STRIG GOSUB). Bij STRIG STOP wordt onderbrekingstoestand aangehouden, d.w.z. dat pas naar de door ON STRIG GOSUB aangegeven subroutine wordt gesprongen als weer de STRIG ON-instructie wordt aangetroffen.

Categorie: instructie

Voorbeeld:
```
10 CLS:ON STRIG GOSUB 40
20 STRIG(0) ON
30 GOTO 30
40 LOCATE 5,5:PRINT "SPATIEBALK INGEDRUKT"
50 FOR I=1 TO 300:NEXT:LOCATE 5,5:PRINT SPC(20)
60 RETURN
```

STR$(<X>)

Geeft een string die de decimale waarde van <X> voorstelt

Categorie: functie

Voorbeeld: `10 A$=STR$(10):PRINT A$`

STRING$(<I>,<J>) of STRING$(<I>,<X$>)

Geeft een string bestaande uit <I> dezelfde schrifttekens die als ASCII-code <J> hebben òf overeenkomen met het eerste schriftteken van <X$>.

Categorie: functie

Voorbeeld: `PRINT STRING$(4,65)`

SWAP <variabele>,<variabele>

Wisselt de waarden van twee variabelen.
Deze variabelen dienen uiteraard wel van hetzelfde type te zijn.

Categorie: instructie

Voorbeeld: `10 A=3:B=5:SWAP A,B:PRINT A,B`

TAB(<X>)

Plaatst de cursor op positie <X> van de regel. Als de cursor reeds voorbij de positie <X> staat, dan gebeurt er niets.
<X> moet liggen tussen 0 en 255. 0 is de meest linkse positie. TAB kan alleen worden gebruikt in een PRINT- of LPRINT-instructie.

Categorie: functie

Voorbeeld: `PRINT TAB(12);"*"`

TAN(<X>)

Geeft de tangens van <X>. <X> moet in radialen worden opgegeven.

Categorie: functie

Voorbeeld: `PRINT TAN(3.1415/8)`

TIME
De systeemvariabele TIME wordt 50 keer per seconde met 1 verhoogd. Deze variabele kunnen we met name gebruiken om werkelijk willekeurige getallen d.m.v. een RND-functie te krijgen (zie aldaar).
Categorie: systeemvariabele
Voorbeeld: `PRINT INT(RND(-TIME)*6+1)`

TROFF
TRON
Met behulp van het commando TRON schakelt men de computer in een toestand waarbij tijdens het uitvoeren van het programma eveneens het regelnummer wordt afgebeeld van de regel waarmee de computer bezig is.
Dit vereenvoudigt het opsporen van fouten. Met TROFF schakelt men deze toestand weer uit.
Categorie: commando
Voorbeeld:
```
10 FOR I=1 TO 3
20 PRINT I
30 NEXT
40 END
TRON
Ok
RUN
[10][20]1
[30][20]2
[30][20]3
[30][40]
Ok
TROFF
Ok
```

USR[<n>](<X>)
Hiermee wijst men een machinetaal-programma aan. <n> is het nummer dat aan het machinetaal-programma is toegekend door DEF USR. Wordt <n> weggelaten dan wordt USR0 verondersteld.
<X> is een argument dat wordt doorgegeven aan het machinetaalprogramma.
<X> wordt op de volgende manier doorgegeven:

1. Alfanumerieke waarden: adres &HF663 bevat de waarde 3. Adressen &HF7F8 en &HF7F9 geven door middel van indirecte adressering waar deze waarde staat. De waarde is in drie bytes opgeslagen: byte 1 bevat de lengte, en byte 2 en 3 bevatten het geheugenadres van de alfanumerieke waarde.
2. Integerwaarden: adres &HF663 bevat de waarde 2. Adressen &HF7F6 tot &HF7F9 bevatten de integerwaarde.
3. Enkele precisiewaarden: adres &HF663 bevat de waarde 4. Adressen &HF7F6 tot &HF7F9 bevatten de enkele precisiewaarde.
4. Dubbele precisie: adres &HF663 bevat de waarde 8. Adressen &HF7F6 tot &HF7FD bevatten de dubbele precisiewaarde.

Waarden die aan MSX BASIC vanuit de subroutine worden teruggemeld, moeten door de machinetaalsubroutine in bovenstaande plaatsen worden opgeslagen. Met CLEAR kunnen we de benodigde ruimte reserveren.

Categorie: functie

Voorbeeld:
```
10 CLEAR 200,&HEFFF
20 AB=&HF000
30 FOR I=AB TO AB+9
40 READ A$:A=VAL("&H"+A$)
50 POKE I,A
60 NEXT I
70 DEFUSR=&HF000
80 INPUT "VOER EEN INTEGER GETAL IN";A%
90 PRINT "INTEGER GETAL=";A%
100 R=USR(A%)
110 PRINT "RESULTAAT IS INTEGER GETAL PLUS 1";R
120 END
130 DATA 23,23,4E,23,46,03,70,2B,71,C9
```

VAL(<X$>)

Geeft de numerieke waarde aan van <X$>. Als het eerste schriftteken van <X$> geen +, −, & of cijfer is, dan wordt VAL gelijk aan 0.

Categorie: functie

Voorbeeld: `PRINT VAL("10")`

VARPTR (<variabele>)

VARPTR (#<X>)

Geeft het adres van de eerste byte van de waarde die hoort bij de <variabele>, dan wel van de eerste byte van een filebesturingsblok.

Aan <variabele> dan wel aan <X> moet een waarde zijn toegekend voordat de functie kan worden opgeroepen.

Categorie: functie

Voorbeeld:
```
10 A=10:B=VARPTR(A)
20 IF B<0 THEN B=B+65536
30 C$="0000"+HEX(B)
40 PRINT RIGHT$(C$,4):END
```

VDP (<X>)

Bevat de inhoud van de VDP-registers. X moet liggen tussen 0 t/m 47, maar niet in het bereik van 25 t/m 32.

Register 8 mag alleen gelezen en nooit beschreven worden.

Categorie: systeemvariabele

Voorbeeld:
```
10 FOR I=0 TO 8
20 A=VDP(I):B$="00000000"+BIN$(A)
30 PRINT RIGHT$(B$,8)
40 NEXT
```

VPEEK (<X>)

Geeft de inhoud van adres <X> van het videogeheugen. X moet liggen tussen 0 en 65535. In SCREEN-modus 5 t/m 8 is het absolute adres gelijk aan X + het startadres van de actieve pagina. Deze pagina volgt uit de volgende tabel:

screen modus	*berekening*
SCREEN-modus 5	paginanummer × &H08000
SCREEN-modus 6	paginanummer × &H08000
SCREEN-modus 7	paginanummer × &H10000
SCREEN-modus 8	paginanummer × &H10000

Categorie: functie

Voorbeeld:
```
10 A=VPEEK(0)
20 A$="00"+HEX$(A)
30 PRINT RIGHT$(A$,2)
```

VPOKE <X, gegeven>

Als POKE, maar nu heeft <X> betrekking op een adres van het video-gedeelte van het RAM-geheugen. Het <X> mag variëren tussen 0 en 65535. Het gegeven mag variëren tussen 0 en 255.

Categorie: instructie

Voorbeeld: `10 VPOKE(0),(VPEEK(0))`

WAIT <poortnummer, uitdrukking 1>[,<uitdrukking 2>]

Heeft betrekking op het inlezen van gegevens via I/O-poort met aangegeven poortnummer.

De ingelezen waarde wordt met <uitdrukking 1> via de zgn. exclusive OR-voorwaarde gecombineerd. Deze uitkomst wordt via een AND-operatie met <uitdrukking 2> gecombineerd. Als het resultaat hiervan 0 oplevert, wordt het inlezen voortgezet en zo niet dan vervolgt de computer het programma met de volgende instructie. Als <uitdrukking 2> wordt weggelaten, wordt de waarde 0 aangenomen.

Categorie: instructie

Voorbeeld: `WAIT &HA8,240`

WIDTH (<aantal kolommen>)

Stelt het aantal kolommen in van het tekstscherm.

In SCREEN-modus 0 kan men 1-80 kolommen instellen. In SCREEN-modus 1 kan men 1-32 kolommen instellen (zie SCREEN).

Bij sommige TV's valt de eerste kolom weg. Het beeld kunt u met SET ADJUST bijstellen! We merken nog het volgende op. Als de computer wordt aangezet zal het beeld volgens SCREEN-modus 0 met een breedte van 37 kolommen worden ingesteld (tenzij dit door middel van SET SCREEN is gewijzigd). Schakelt men over naar SCREEN-modus 1 dan zal het beeld in 29 kolommen zijn ingedeeld.

Categorie: instructie

Voorbeeld: `SCREEN 0:WIDTH 80`

INDEX

MSX2-BASIC is een logisch vervolg op het inmiddels wijd verbreide MSX-BASIC, dat zich tot een echte standaard heeft ontwikkeld.
In dit boek worden de, voor de beginner belangrijke, principes van MSX-BASIC beschreven. Bovendien worden de verschillen met MSX2-BASIC afzonderlijk onder de loep genomen. Daarbij is het natuurlijk belangrijk, dat MSX-BASIC-programma's ook in MSX2-BASIC te gebruiken zijn.
In een aantal afzonderlijke appendices komt de omgang met de randapparatuur aan de orde, waarbij de ingebouwde diskdrive een belangrijke plaats inneemt.
Alle MSX2-BASIC-opdrachten worden, in alfabetische volgorde, beschreven. Elke opdracht wordt geïllustreerd met een kort voorbeeld-programma waardoor ook voor de beginnende MSX-programmeur de opdrachten snel toepasbaar zijn.
Ten slotte is voor de meer gevorderde programmeur een aparte appendix opgenomen met monitorgegevens.
Diegenen, die al enige ervaring hebben in het programmeren in Z80-assembly vinden hierin een korte beschrijving van de BIOS-routines.
Al met al is dit boek hiermee het meest complete handboek, dat voor de MSX2-computer beschikbaar is.

ISBN 90 201 1961 3